

Analiza danych z pakietem R

Kurs wprowadzający

Paweł Lula, Katedra Systemów Obliczeniowych,
Uniwersytet Ekonomiczny w Krakowie
pawel.lula@uek.krakow.pl

Informacje organizacyjne

- Kurs „Analiza danych z pakietem R”
 - Wprowadzenie. Struktury danych. Funkcje. Podstawy programowania (6 godzin, Paweł Lula)
 - Przygotowanie oraz wizualizacja danych. Wprowadzenie do analizy danych (6 godzin, Przemysław Jaśko)
 - Wprowadzenie do wnioskowania statystycznego (4 godziny, Daniel Kosiorowski)
 - Elementy uczenia maszynowego (analiza skupień, klasyfikacja) (2 godziny, Paweł Lula)

Informacje organizacyjne


- Kursy tematyczne:
 - Programowanie w języku R,
 - Eksploracyjna analiza danych tekstowych,
 - Elementy statystyki odpornej,
 - Analiza sieci społecznych,
 - Analiza szeregów czasowych,
 - Sieci neuronowe,
 - Metody optymalizacji,
 - Dydaktyka i tworzenie publikacji z pakietem R,
 - *inne...*

Materiały do kursu

- Platforma Moodle
- <https://e-uczelnia.uek.krakow.pl/course/view.php?id=6921>
- Hasło dostępu: r4

R - <http://www.r-project.org/>

Bezpieczna | <https://www.r-project.org>



[Home]
Download
 CRAN
R Project
 About R
 Logo
 Contribution

The R Project for Statistical Computing

Getting Started

R is a free software environment for statistical computing and graphics. It compiles and runs on a wide variety of UNIX platforms, Windows and MacOS. To [download R](#), please choose your preferred [CRAN mirror](#).

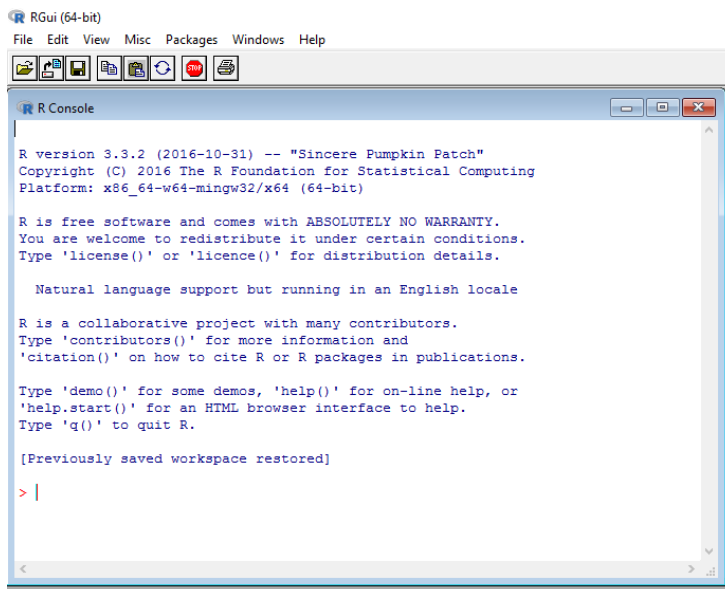
If you have questions about R like how to download and install the software, or what the license terms are, please read our [answers to frequently asked questions](#) before you send an email.

News

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

5

R Console



```
RGui (64-bit)
File Edit View Misc Packages Windows Help

R Console

R version 3.3.2 (2016-10-31) -- "Sincere Pumpkin Patch"
Copyright (C) 2016 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

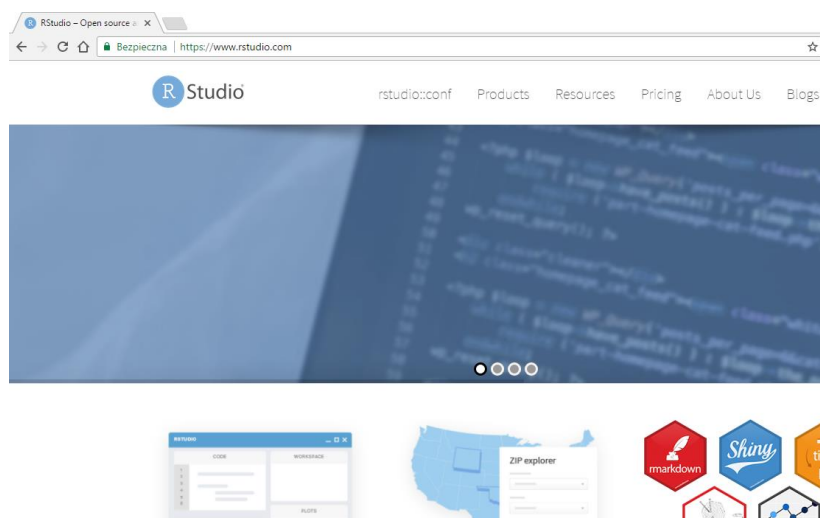
[Previously saved workspace restored]

> |
```

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

6

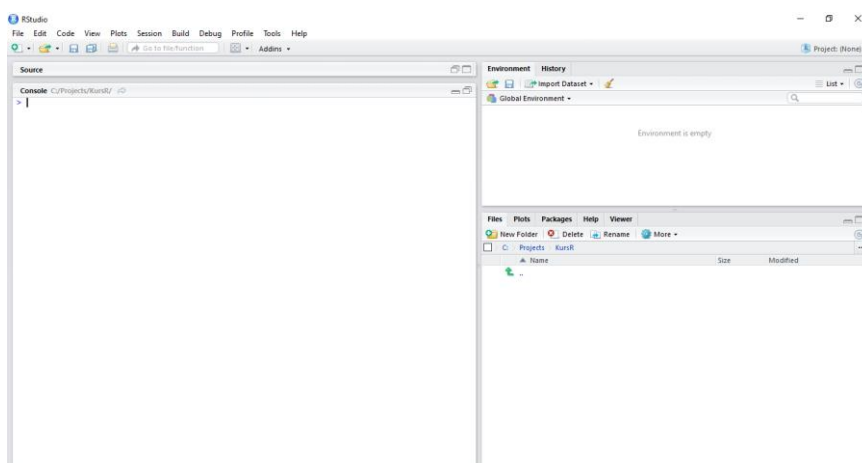
<https://www.rstudio.com/>



Paweł Lula, Katedra Systemów Obliczeniowych, UEK

7

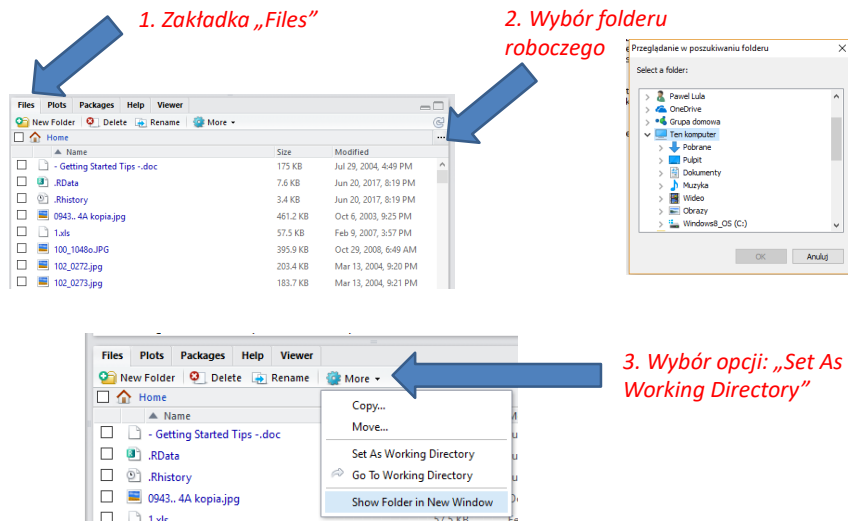
RStudio



Paweł Lula, Katedra Systemów Obliczeniowych, UEK

8

RStudio – określenie folderu roboczego



Paweł Lula, Katedra Systemów Obliczeniowych, UEK

9

Proste obliczenia w pakiecie R

```
> 2+7
[1] 9
> (1 + 3) / 8
[1] 0.5
> pi
[1] 3.141593
> abs(-2)
[1] 2
> 2^4
[1] 16
> 2**4
[1] 16
> 1/3
[1] 0.3333333
>
```

← wartość π

← potęgowanie (2 do potęgi 4)

← potęgowanie (2 do potęgi 4)

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

10

Proste obliczenia w pakiecie R

```

> sin(30)           ← funkcja sinus, argument w radianach
[1] -0.9880316
> sin(30*pi/180)    ← funkcja sinus (zamiana stopni na radiany)
[1] 0.5
> log(2)            ← logarytm naturalny
[1] 0.6931472
> log10(10)         ← logarytm przy podstawie dziesięć
[1] 1
> log2(2)           ← logarytm przy podstawie 2
[1] 1
>

```

Proste obliczenia w pakiecie R

```

> log(2)            ← logarytm naturalny
[1] 0.6931472
> log(25,5)         ← logarytm z 25 przy podstawie 5
[1] 2
> log(10)           ← logarytm naturalny
[1] 2.302585
> log(10,2)         ← logarytm z 10 przy podstawie 2
[1] 3.321928
> 2**log(10,2)      ← 2 do potęgi log(10,2)
[1] 10
> 2^log(10,2)       ← 2 do potęgi log(10,2)
[1] 10
>

```

Proste obliczenia w pakiecie R

```
> exp(1)           ← funkcja wykładnicza
[1] 2.718282
> log(38)          ← logarytm naturalny
[1] 3.637586
> exp(log(38))
[1] 38
>
```

Zmienne i instrukcja przypisania

```
> x<-4             ← instrukcja przypisania
> x                ← wyświetlenie wartości zmiennej
[1] 4
> x=5              ← instrukcja przypisania (postać niezalecana)
> x
[1] 5
> ls()             ← lista obiektów w przestrzeni roboczej
[1] "x"
> rm(s)            ← usunięcie obiektu s
> rm(list=ls())    ← usuwanie wszystkich zmiennych z
                    przestrzeni roboczej

> ls()
character(0)
>
```

Zapis i odczyt obiektów. Instrukcje „save” i „load”

```

> ls()
character(0)
> x <- 1
> y <- 2
> z <- 3
> ls()
[1] "x" "y" "z"
> save(x,y,file="data1.RData")  ← zapis do pliku wskazanych obiektów
> rm(list=ls())
> ls()
character(0)
> load(file="data1.RData")      ← odczyt obiektów z pliku
> ls()
[1] "x" "y"
> x
[1] 1
> y
[1] 2
>

```

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

15

Wyrażenia arytmetyczne

```

> 16+3
[1] 19
> 16-3
[1] 13
> 16/3
[1] 5.333333
> 16*3
[1] 48
> 16^3
[1] 4096
> 16**3
[1] 4096
> 16%%3  ← reszta z dzielenia
[1] 1
> 16%/%3  ← dzielenie całkowite
[1] 5
> (16-3)/(32-5)
[1] 0.4814815
>

```

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

16

Wartości i operatory logiczne

```

> x<-TRUE
> y<-FALSE
> !x           ← operator negacji
[1] FALSE
> !y
[1] TRUE
> x|y          ← operator alternatywy (OR)
[1] TRUE
> x&y          ← operator koniunkcji (AND)
[1] FALSE
> x&!y
[1] TRUE
> isTRUE(x)
[1] TRUE
> isTRUE(y)
[1] FALSE
>

```

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

17

Operatory według malejącego priorytetu

:: :::	access variables in a namespace
\$ @	component / slot extraction
[[[indexing
^	exponentiation (right to left)
- +	unary minus and plus
:	sequence operator
%any%	special operators (including %% and %/%)
* /	multiply, divide
+ -	(binary) add, subtract
< > <= >= == !=	ordering and comparison
!	negation
& &&	and
	or
~	as in formulae
-> ->>	rightwards assignment
<- <<-	assignment (right to left)
=	assignment (right to left)
?	help (unary and binary)

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

18

Operatory relacyjne

```

> x<-3
> y<-8
> x>y
[1] FALSE
> x<y
[1] TRUE
> x>=y
[1] FALSE
> x<=y
[1] TRUE
> x==y          ← czy x jest równe y?
[1] FALSE
> x!=y          ← czy x jest różne od y?
[1] TRUE
>

```

Łańcuchy znaków

```

> x<-"abcdef"
> x
[1] "abcdef"
> cat(x)          ← wyświetlenie wartości
abcdef
> x<-"abc\ndef"   ← \n oznacza przejście do nowego wiersza
> x
[1] "abc\ndef"
> cat(x)
abc
def
> nchar(x)        ← liczba znaków w łańcuchu
[1] 7
> x<-"a\tb\tc\nd\te\tf"  ← \t oznacza znak tabulacji
> x
[1] "a\tb\tc\nd\te\tf"
> cat(x)
a      b      c
d      e      f
>

```

łańcuchy znaków

```

> x <- "wybierz \"tak\" lub \"nie\""
> x
[1] "wybierz \"tak\" lub \"nie\""
> cat(x)
wybierz "tak" lub "nie"
> paste("abc","def","ghi")
[1] "abc def ghi"
> paste("abc","def","ghi",sep="*")
[1] "abc*def*ghi"
> paste("abc","def","ghi",sep="")
[1] "abcdefghi"
>

```

← \" oznacza cudzysłów

← łączenie łańcuchów

łańcuchy znaków

```

> x<-"Analiza danych"
> tolower(x)
[1] "analiza danych"
> toupper(x)
[1] "ANALIZA DANYCH"
> x<-"Łódź"
> tolower(x)
[1] "łódź"
> toupper(x)
[1] "ŁÓDŹ"
>

```

Określenie typu wartości

```

> x<-"abc"
> y<-5.7
> z<-TRUE
> typeof(x)           ← zwraca informację o typie obiektu
[1] "character"
> typeof(y)
[1] "double"
> typeof(z)
[1] "logical"
> typeof(4+4)
[1] "double"
> typeof(4>3)
[1] "logical"
> typeof(4=3)         ← to nie jest sprawdzenie równości!
Error: unexpected '=' in "typeof(4="
> typeof(4==3)
[1] "logical"
>

```

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

23

Zmiana typu wartości

```

> x<-"abc"
> y<-5.7
> z<-TRUE
> is.numeric(x)
[1] FALSE
> is.numeric(y)
[1] TRUE
> is.character(x)
[1] TRUE
> is.character(y)
[1] FALSE
> as.character(y)
[1] "5.7"
>

```

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

24

Zmiana typu wartości

```
> as.numeric(TRUE)
[1] 1
> as.numeric(FALSE)
[1] 0
> as.logical(-1)
[1] TRUE
> as.logical(0)
[1] FALSE
> as.logical(1)
[1] TRUE
> as.logical(10)
[1] TRUE
>
```

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

25

Braki danych

```
> x<-NA           ← NA - not available
> x
[1] NA
> 2*x
[1] NA
> cat(x)
NA
> is.na(x)
[1] TRUE
> x<-4
> is.na(x)
[1] FALSE
>
```

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

26

WEKTORY

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

27

Wektory

- Wektor – struktura danych złożona z elementów identyfikowanych za pomocą indeksów

```
> x <- c(10, 20, 30, 40)    ← tworzenie wektora
> x
[1] 10 20 30 40
> x[1]
[1] 10
> x[4]
[1] 40
> x[5]
[1] NA
```

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

28

Wektory

```
> x<-1:10
> x
[1] 1 2 3 4 5 6 7 8 9 10
> x<-1:250
> x
[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
[21] 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
[41] 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60
[61] 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80
[81] 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
[101] 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120
[121] 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140
[141] 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160
[161] 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180
[181] 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200
[201] 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220
[221] 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240
[241] 241 242 243 244 245 246 247 248 249 250
>
```

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

29

Wektory

```
> x<-seq(1,12)
> x
[1] 1 2 3 4 5 6 7 8 9 10 11 12
> x<-seq(from=1,to=5)
> x
[1] 1 2 3 4 5
> x<-seq(to=1,from=5)
> x
[1] 5 4 3 2 1
> x<-seq(from=1,to=5,by=1.2)
> x
[1] 1.0 2.2 3.4 4.6
> x<-seq(from=1,by=1.2,length=12)
> x
[1] 1.0 2.2 3.4 4.6 5.8 7.0 8.2 9.4 10.6 11.8 13.0 14.2
>
```

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

30

Wektory

```
> rep(c(1,2,3),times=3)
[1] 1 2 3 1 2 3 1 2 3
> rep(c(1,2,3),each=3)
[1] 1 1 1 2 2 2 3 3 3
> x<-seq(from=10,to=60,by=10)
> x
[1] 10 20 30 40 50 60
> x[2]
[1] 20
> x[c(1,2,4)]
[1] 10 20 40
> x[c(1,2,4,4,4)]
[1] 10 20 40 40 40
```

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

31

Wektory

```
> x
[1] 10 20 30 40 50 60
> x.log <- c(rep(TRUE,3),rep(FALSE,3))
> x.log
[1] TRUE TRUE TRUE FALSE FALSE FALSE
> x[x.log]
[1] 10 20 30
> x[x >= 20]
[1] 20 30 40 50 60
> x[x >= 20 & x <= 40]
[1] 20 30 40
> x
[1] 10 20 30 40 50 60
> x[-c(1,4)]
[1] 20 30 50 60
```

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

32

Wektory

```
> x<-seq(from=10,to=60,by=10)
> length(x)                ← liczba elementów w wektorze
[1] 6
> 1:length(x)
[1] 1 2 3 4 5 6
> 1:length(x)%%2
[1] 1 0 1 0 1 0
> 1:length(x)%%2==0
[1] FALSE  TRUE FALSE  TRUE FALSE  TRUE
> x[1:length(x)%%2==0]     ← tylko elementy na parzystych
                             pozycjach
[1] 20 40 60
>
```

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

33

Wektory

```
> x<-seq(from=1,to=10,by=0.1)
> x
[1] 1.0 1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 2.0 2.1 2.2 2.3 2.4 2.5
[17] 2.6 2.7 2.8 2.9 3.0 3.1 3.2 3.3 3.4 3.5 3.6 3.7 3.8 3.9 4.0 4.1
[33] 4.2 4.3 4.4 4.5 4.6 4.7 4.8 4.9 5.0 5.1 5.2 5.3 5.4 5.5 5.6 5.7
[49] 5.8 5.9 6.0 6.1 6.2 6.3 6.4 6.5 6.6 6.7 6.8 6.9 7.0 7.1 7.2 7.3
[65] 7.4 7.5 7.6 7.7 7.8 7.9 8.0 8.1 8.2 8.3 8.4 8.5 8.6 8.7 8.8 8.9
[81] 9.0 9.1 9.2 9.3 9.4 9.5 9.6 9.7 9.8 9.9 10.0
> x[x%%3==0]               ← wyszukaj elementy podzielne przez 3
[1] 3 6 9
> which(x%%3==0)           ← podaj pozycje elementów podzielnych przez 3
[1] 21 51 81
>
```

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

34

Wektory

```

> x<-seq(from=-2.5,to=2.5,by=0.5)
> x
[1] -2.5 -2.0 -1.5 -1.0 -0.5  0.0  0.5  1.0  1.5  2.0  2.5
> abs(x)           ← wartość bezwzględna
[1] 2.5 2.0 1.5 1.0 0.5 0.0 0.5 1.0 1.5 2.0 2.5
> sign(x)          ← znak
[1] -1 -1 -1 -1 -1  0  1  1  1  1  1
> floor(x)         ← zaokrąglenie w dół
[1] -3 -2 -2 -1 -1  0  0  1  1  2  2
> ceiling(x)       ← zaokrąglenie w górę
[1] -2 -2 -1 -1  0  0  1  1  2  2  3
> trunc(x)         ← zaokrąglenie w kierunku zera
[1] -2 -2 -1 -1  0  0  0  1  1  2  2
> round(x)         ← zaokrąglenie w kierunku najbliższej parzystej
[1] -2 -2 -2 -1  0  0  0  1  2  2  2
>

```

round(1.5) = 2
część całkowita nieparzysta → w górę

round(2.5) = 2
część całkowita parzysta → w dół

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

35

Wektory

```

> y<-x/0.7632
> y
[1] -3.2756813 -2.6205451 -1.9654088 -1.3102725 -0.6551363  0.0000000  0.6551363
[8]  1.3102725  1.9654088  2.6205451  3.2756813
> round(y,2)      ← zaokrąglenie do dwóch miejsc po przecinku
[1] -3.28 -2.62 -1.97 -1.31 -0.66  0.00  0.66  1.31  1.97  2.62  3.28
>

```

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

36

Wektory

```
> p
[1] 4470 4096 4169 4419 4143 3351 3338 4178 3492 3037 4427 4808 4329 4335 3854 3990
[17] 4295 4268 4404 4701 4078 3812 4283 3809 3596 4603 4794 4360 3798 4086 3896 3983
[33] 4669 3695 4216 4179 4012 4690 3513 4861 3073 4429 4002 4124 2476 4341 4328 4017
[49] 3956 3840 4606 4243 3241 3210 4264 4867 4049 3756 3688 4562 3801 4362 3677 4012
[65] 3710 3258 4458 3551 3998 4179 3762 4164 2922 3603 3671 3315 3835 3971 2978 5259
[81] 4279 3848 3949 3802 4218 3559 3802 4063 3698 4002 3308 4259 3671 4097 3650 4170
[97] 4084 3595 4235 4253
> signif(p,2)      ← zaokrąglenie do dwóch cyfr znaczących
[1] 4500 4100 4200 4400 4100 3400 3300 4200 3500 3000 4400 4800 4300 4300 3900 4000
[17] 4300 4300 4400 4700 4100 3800 4300 3800 3600 4600 4800 4400 3800 4100 3900 4000
[33] 4700 3700 4200 4200 4000 4700 3500 4900 3100 4400 4000 4100 2500 4300 4300 4000
[49] 4000 3800 4600 4200 3200 3200 4300 4900 4000 3800 3700 4600 3800 4400 3700 4000
[65] 3700 3300 4500 3600 4000 4200 3800 4200 2900 3600 3700 3300 3800 4000 3000 5300
[81] 4300 3800 3900 3800 4200 3600 3800 4100 3700 4000 3300 4300 3700 4100 3600 4200
[97] 4100 3600 4200 4300
> signif(p,3)      ← zaokrąglenie do trzech cyfr znaczących
[1] 4470 4100 4170 4420 4140 3350 3340 4180 3490 3040 4430 4810 4330 4340 3850 3990
[17] 4300 4270 4400 4700 4080 3810 4280 3810 3600 4600 4790 4360 3800 4090 3900 3980
[33] 4670 3700 4220 4180 4010 4690 3510 4860 3070 4430 4000 4120 2480 4340 4330 4020
[49] 3960 3840 4610 4240 3240 3210 4260 4870 4050 3760 3690 4560 3800 4360 3680 4010
[65] 3710 3260 4460 3550 4000 4180 3760 4160 2920 3600 3670 3320 3840 3970 2980 5260
[81] 4280 3850 3950 3800 4220 3560 3800 4060 3700 4000 3310 4260 3670 4100 3650 4170
[97] 4080 3600 4240 4250
>
```

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

37

Wektory

```
> p
[1] 4470 4096 4169 4419 4143 3351 3338 4178 3492 3037 4427 4808 4329 4335 3854 3990
[17] 4295 4268 4404 4701 4078 3812 4283 3809 3596 4603 4794 4360 3798 4086 3896 3983
[33] 4669 3695 4216 4179 4012 4690 3513 4861 3073 4429 4002 4124 2476 4341 4328 4017
[49] 3956 3840 4606 4243 3241 3210 4264 4867 4049 3756 3688 4562 3801 4362 3677 4012
[65] 3710 3258 4458 3551 3998 4179 3762 4164 2922 3603 3671 3315 3835 3971 2978 5259
[81] 4279 3848 3949 3802 4218 3559 3802 4063 3698 4002 3308 4259 3671 4097 3650 4170
[97] 4084 3595 4235 4253
> round(p,-2)      ← zaokrąglenie do pełnych setek
[1] 4500 4100 4200 4400 4100 3400 3300 4200 3500 3000 4400 4800 4300 4300 3900 4000
[17] 4300 4300 4400 4700 4100 3800 4300 3800 3600 4600 4800 4400 3800 4100 3900 4000
[33] 4700 3700 4200 4200 4000 4700 3500 4900 3100 4400 4000 4100 2500 4300 4300 4000
[49] 4000 3800 4600 4200 3200 3200 4300 4900 4000 3800 3700 4600 3800 4400 3700 4000
[65] 3700 3300 4500 3600 4000 4200 3800 4200 2900 3600 3700 3300 3800 4000 3000 5300
[81] 4300 3800 3900 3800 4200 3600 3800 4100 3700 4000 3300 4300 3700 4100 3600 4200
[97] 4100 3600 4200 4300
> round(p,-1)      ← zaokrąglenie do pełnych dziesiątek
[1] 4470 4100 4170 4420 4140 3350 3340 4180 3490 3040 4430 4810 4330 4340 3850 3990
[17] 4300 4270 4400 4700 4080 3810 4280 3810 3600 4600 4790 4360 3800 4090 3900 3980
[33] 4670 3700 4220 4180 4010 4690 3510 4860 3070 4430 4000 4120 2480 4340 4330 4020
[49] 3960 3840 4610 4240 3240 3210 4260 4870 4050 3760 3690 4560 3800 4360 3680 4010
[65] 3710 3260 4460 3550 4000 4180 3760 4160 2920 3600 3670 3320 3840 3970 2980 5260
[81] 4280 3850 3950 3800 4220 3560 3800 4060 3700 4000 3310 4260 3670 4100 3650 4170
[97] 4080 3600 4240 4250
> round(p,-3)      ← zaokrąglenie do pełnych tysięcy
[1] 4000 4000 4000 4000 4000 3000 3000 4000 3000 3000 4000 5000 4000 4000 4000 4000
[17] 4000 4000 4000 5000 4000 4000 4000 4000 5000 5000 4000 4000 4000 4000 4000 4000
[33] 5000 4000 4000 4000 5000 4000 4000 5000 3000 4000 4000 2000 4000 4000 4000 4000
[49] 4000 4000 5000 4000 3000 3000 4000 5000 4000 4000 5000 4000 4000 4000 4000 4000
[65] 4000 3000 4000 4000 4000 4000 4000 4000 3000 4000 4000 3000 4000 4000 3000 5000
[81] 4000 4000 4000 4000 4000 4000 4000 4000 4000 4000 3000 4000 4000 4000 4000 4000
[97] 4000 4000 4000 4000
```

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

38

Wartości pseudolosowe

```
> runif(1,min=0,max=1)      ← wartości losowe o rozkładzie równomiernym
[1] 0.546232
> runif(25,min=0,max=1)
[1] 0.22322398 0.87020706 0.41534248 0.01003917 0.13747933 0.81112089 0.56669522
[8] 0.72175324 0.76296297 0.19471121 0.40233781 0.22580990 0.86431986 0.75944655
[15] 0.22766822 0.19354859 0.58777844 0.87447204 0.14979593 0.16596207 0.29823006
[22] 0.16295145 0.98235618 0.52938952 0.70306195
> runif(25)
[1] 0.7395029 0.5667565 0.7204415 0.1146010 0.1477919 0.9613592 0.3497272 0.8269492
[9] 0.2071194 0.6785840 0.9949426 0.7580600 0.2935716 0.1532010 0.2803508 0.2845353
[17] 0.5641698 0.2857825 0.8829239 0.2873873 0.7340149 0.7590553 0.4015707 0.8656241
[25] 0.9212422
> runif(25,min=10,max=12)
[1] 11.24184 10.04960 10.27392 10.74141 11.84720 10.13371 10.83743 11.02469 11.87519
[10] 11.93820 10.04985 10.08369 10.30626 10.09209 11.76965 10.38350 10.93830 10.17015
[19] 11.51776 10.05132 11.04544 10.06008 11.01472 10.45574 10.28292
>
```

Wartość początkowa dla generatora liczb pseudolosowych

```
> set.seed(123)             ← wartość początkowa (ziarno)
> runif(25,min=0,max=1)
[1] 0.28757752 0.78830514 0.40897692 0.88301740 0.94046728 0.04555650 0.52810549
[8] 0.89241904 0.55143501 0.45661474 0.95683335 0.45333416 0.67757064 0.57263340
[15] 0.10292468 0.89982497 0.24608773 0.04205953 0.32792072 0.95450365 0.88953932
[22] 0.69280341 0.64050681 0.99426978 0.65570580
> set.seed(123)
> runif(25,min=0,max=1)
[1] 0.28757752 0.78830514 0.40897692 0.88301740 0.94046728 0.04555650 0.52810549
[8] 0.89241904 0.55143501 0.45661474 0.95683335 0.45333416 0.67757064 0.57263340
[15] 0.10292468 0.89982497 0.24608773 0.04205953 0.32792072 0.95450365 0.88953932
[22] 0.69280341 0.64050681 0.99426978 0.65570580
>
```

Wartości pseudolosowe

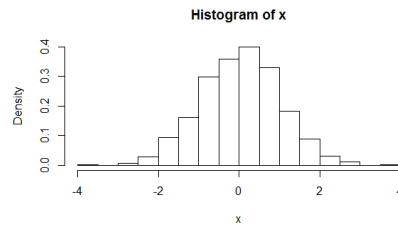
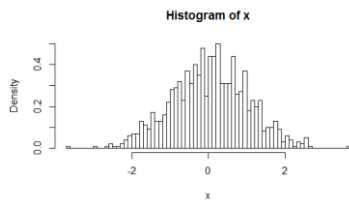
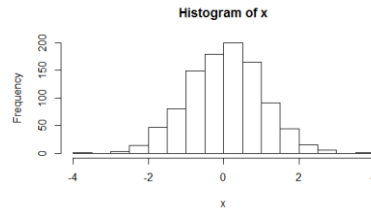
```
> rnorm(1)           ← wartości losowe o rozkładzie normalnym
[1] 0.5862529
> rnorm(1,mean=0,sd=1)
[1] 1.493466
> rnorm(10,mean=0,sd=1)
[1] 0.65173788 -0.09922869 -0.22125674 0.87321824 0.54161623 0.89082284
[7] 1.54449212 -1.54723752 -0.09058480 1.14255552
> rnorm(10,mean=4,sd=5)
[1] 7.2258009 -2.7249680 1.9502428 8.4838285 6.1435551 0.4355262 2.3694181
[8] 2.4029528 -3.0216259 -8.9696823
>
```

Losowanie wartości z wektora

```
> kostka<-1:6
> kostka
[1] 1 2 3 4 5 6
> sample(kostka)           ← losowanie sześciu wartości bez zwracania (permutacja)
[1] 5 2 1 6 4 3
> sample(kostka,5)        ← losowanie pięciu wartości bez zwracania
[1] 3 6 1 2 5
> sample(kostka,5)
[1] 3 5 2 4 1
> sample(kostka,7)
Error in sample.int(length(x), size, replace, prob) :
  cannot take a sample larger than the population when 'replace = FALSE'
> sample(kostka,5,replace=TRUE) ← losowanie ze zwracaniem
[1] 6 6 4 3 5
> sample(kostka,50,replace=TRUE) ← losowanie ze zwracaniem
[1] 1 4 2 3 2 3 5 3 1 2 4 2 2 5 5 4 2 1 2 1 5 3 6 2 6 1 6 2 2 1 5 4 2 1 1 2 1 2 2 4
[41] 4 3 4 3 5 4 5 6 2 3
> sample(kostka,50,replace=FALSE) ← !!!
Error in sample.int(length(x), size, replace, prob) :
  cannot take a sample larger than the population when 'replace = FALSE'
>
```

Histogram

```
> x<-rnorm(1000)
> hist(x)
> hist(x,probability=TRUE)
> hist(x,probability=TRUE,breaks=100)
>
```



Paweł Lula, Katedra Systemów Obliczeniowych, UEK

43

Funkcje do działań na wektorach

```
> x<-rnorm(1000)
> sum(x)           ← suma
[1] -14.66292
> prod(x)          ← iloczyn
[1] -1.120001e-270
> mean(x)          ← średnia
[1] -0.01466292
> var(x)           ← wariancja z próby
[1] 0.9990732
> sd(x)            ← odchylenie standardowe
[1] 0.9995365
> min(x)
[1] -2.732332
> max(x)
[1] 4.078857
> median(x)
[1] 0.007368256
```

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

44

Funkcje do działań na wektorach

```
> x<-rnorm(1000)
> median(x)
[1] 0.007368256
> quantile(x,0.5)      ← kwantyle
      50%
0.007368256
> quantile(x,0.25)
      25%
-0.725015
> quantile(x,0.75)
      75%
0.672103
> quantile(x,0.7123)
      71.23%
0.5377364
> which(x>quantile(x,0.99))
[1] 178 308 329 344 407 551 578 637 856 961
> x[x>quantile(x,0.99)]
[1] 2.813945 2.614690 4.078857 2.650450 2.488869 2.465143 2.973285 2.370028 2.341749
[10] 3.068313
>
```

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

45

Analiza zawartości wektora

```
> x<-rnorm(1000)
> any(x<2)      ← czy jest w wektorze przynajmniej jedna wartość
[1] TRUE
> all(x<2)      ← czy wszystkie wartości w wektorze spełniają warunek
[1] FALSE
> all(x<5)
[1] TRUE
> min(x)        ← wartość minimalna
[1] -3.899816
> which.min(x)  ← pozycja wartości minimalnej
[1] 175
> max(x)        ← wartość maksymalna
[1] 2.741868
> which.max(x)  ← pozycja wartości maksymalnej
[1] 860
>
```

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

46

Sortowanie

```
> x<-runif(25)
> x
[1] 0.29343869 0.93404167 0.25421634 0.69956300 0.78686031 0.13667907 0.09921396
[8] 0.72454105 0.92512723 0.14738371 0.20500078 0.45022089 0.06131211 0.53353635
[15] 0.62622036 0.84820761 0.28833917 0.24312796 0.24162267 0.71865470 0.25891355
[22] 0.91854926 0.90673777 0.55600510 0.15775449
> sort(x)                                ← sortowanie /wartości rosnąco/
[1] 0.06131211 0.09921396 0.13667907 0.14738371 0.15775449 0.20500078 0.24162267
[8] 0.24312796 0.25421634 0.25891355 0.28833917 0.29343869 0.45022089 0.53353635
[15] 0.55600510 0.62622036 0.69956300 0.71865470 0.72454105 0.78686031 0.84820761
[22] 0.90673777 0.91854926 0.92512723 0.93404167
> sort(x,decreasing=TRUE)                ← sortowanie /wartości malejąco/
[1] 0.93404167 0.92512723 0.91854926 0.90673777 0.84820761 0.78686031 0.72454105
[8] 0.71865470 0.69956300 0.62622036 0.55600510 0.53353635 0.45022089 0.29343869
[15] 0.28833917 0.25891355 0.25421634 0.24312796 0.24162267 0.20500078 0.15775449
[22] 0.14738371 0.13667907 0.09921396 0.06131211
> order(x)                              ← indeksy elementów uporządkowanych rosnąco
[1] 13 7 6 10 25 11 19 18 3 21 17 1 12 14 24 15 4 20 8 5 16 23 22 9 2
> order(x,decreasing=TRUE)               ← indeksy elementów uporządkowanych malejąco
[1] 2 9 22 23 16 5 8 20 4 15 24 14 12 1 17 21 3 18 19 11 25 10 6 7 13
> x<-sort(x)                             ← zapamiętanie posortowanego wektora
>
```

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

47

Sortowanie

```
> x<-runif(10)
> x
[1] 0.3199548 0.2831284 0.1310351 0.1731114 0.1525631 0.5423266 0.7786321 0.7513898
[9] 0.4160406 0.2255702
> sort(x)                                ← sortowanie
[1] 0.1310351 0.1525631 0.1731114 0.2255702 0.2831284 0.3199548 0.4160406 0.5423266
[9] 0.7513898 0.7786321
> order(x)                              ← indeksy uporządkowanych wartości
[1] 3 5 4 10 2 1 9 6 8 7
> rank(x)                               ← ranking wartości
[1] 6 5 1 3 2 8 10 9 7 4
>
```

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

48

Przykładowe uporządkowanie

```

> wzrost<-c(167,180,152,191,120)
> waga<-c(78,90,45,88,25)
> sort(waga)
[1] 25 45 78 88 90
> sort(wzrost)
[1] 120 152 167 180 191
> order(waga)
[1] 5 3 1 4 2
> wzrost[order(waga)]      ← uszeregowanie wzrostu w sposób zapewniający
                             rosnące uporządkowanie wagi
[1] 120 152 167 191 180
> wzrost[order(waga,decreasing=TRUE)]
[1] 180 191 167 152 120
>

```

Wektory

```

> s<-c("jeden","dwa","trzy")
> s
[1] "jeden" "dwa"   "trzy"
> s[2]
[1] "dwa"

```

Wektory

```
> x<-c(1,2,3)
> y<-c(6,7,8)
> x+y      ← dodawanie odpowiadających sobie elementów
[1]  7  9 11
> x*y      ← mnożenie odpowiadających sobie elementów
[1]  6 14 24
```

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

51

Wektory

```
> x<-c(1,2,3)
> z<-c(1,2,3,4,5)
> x+z
[1] 2 4 6 5 7
Warning message:
  longer object length is not a multiple of shorter object
  length in: x + z
>
```

x	1	2	3	1	2	3
z	1	2	3	4	5	
x+z	2	4	6	5	7	

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

52

Wektory

```

> x<-c(1,2,3)
> y<-c(3,4,2)
> x
[1] 1 2 3
> y
[1] 3 4 2
> x*y          ← mnożenie odpowiadających sobie elementów
[1] 3 8 6
> x%*%y        ← iloczyn skalarny wektorów
      [,1]
[1,]    17

```

1	2	3
---	---	---

3
4
2

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

53

Wektory

```

> x<-c(1,2,3)
> y<-c(3,4,2)
> x%*%y          ← iloczyn skalarny, iloczyn wewnętrzny
      [,1]
[1,]    17
> x%o%y          ← iloczyn zewnętrzny
      [,1] [,2] [,3]
[1,]    3    4    2
[2,]    6    8    4
[3,]    9   12    6
> outer(x,y)     ← iloczyn zewnętrzny
      [,1] [,2] [,3]
[1,]    3    4    2
[2,]    6    8    4
[3,]    9   12    6

```

1			
2	3	4	2
3			

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

54

Działania logiczne na wektorach

```
> c(FALSE, FALSE, TRUE) & c(TRUE, FALSE, TRUE)  ← dla każdego elementu
[1] FALSE FALSE TRUE

> c(FALSE, FALSE, TRUE) && c(TRUE, FALSE, TRUE) ← dla całego wektora
[1] FALSE

> c(FALSE, FALSE, TRUE) | c(TRUE, FALSE, TRUE)
[1] TRUE FALSE TRUE

> c(FALSE, FALSE, TRUE) || c(TRUE, FALSE, TRUE)
[1] TRUE

> c(FALSE, FALSE, FALSE) | c(FALSE, FALSE, FALSE)
[1] FALSE FALSE FALSE

> c(FALSE, FALSE, FALSE) || c(FALSE, FALSE, FALSE)
[1] FALSE
>
```

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

55

Zbiory

Elementy zbiorów przechowywane są w postaci wektorów o niepowtarzających się wartościach

```
> x<-c(1,4,12)
> y<-4:15
> union(x,y)      ← suma zbiorów
[1] 1 4 12 5 6 7 8 9 10 11 13 14 15
> intersect(x,y)  ← iloczyn zbiorów
[1] 4 12
> setdiff(x,y)    ← różnica zbiorów
[1] 1
> setdiff(y,x)    ← różnica zbiorów
[1] 5 6 7 8 9 10 11 13 14 15
>
```

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

56

Zbiory

```

> x<-c(1,2,3,4)
> y<-c(3,4,1,2)
> x==y           ← porównywanie odpowiadających sobie elementów
[1] FALSE FALSE FALSE FALSE
> setequal(x,y)  ← porównywanie zbiorów
[1] TRUE
> is.element(3,x) ← sprawdzanie przynależności do zbioru
[1] TRUE
> is.element(8,x)
[1] FALSE
> 3 %in% x       ← sprawdzanie przynależności do zbioru
[1] TRUE
> 8 %in% x
[1] FALSE
>

```

Zbiory

```

> x<-c(2,5,3,1,6)
> y<-c(3,6,2,3,3)
> duplicated(x)    ← czy wartości się nie powtarzają?
[1] FALSE FALSE FALSE FALSE FALSE
> duplicated(y)
[1] FALSE FALSE FALSE TRUE TRUE
> which(duplicated(y)) ← które wartości się powtarzają?
[1] 4 5
> unique(y)       ← zbiór niepowtarzających się wartości
[1] 3 6 2
>

```

Macierze

- `matrix(data = NA, nrow = 1, ncol = 1, byrow = FALSE, dimnames = NULL)`
- Parametry:
 - `data` – wektor elementów przeznaczonych do umieszczenia w macierzy
 - `nrow` – liczba wierszy
 - `ncol` – liczba kolumn
 - `byrow` – sposób rozmieszczenia elementów w macierzy
 - `dimnames` – nazwy kolumn

Macierze

```
> x<-matrix(c(1,2,3,4,5,6,7,8,9),3,3)
> x
      [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
> x<-matrix(c(1,2,3,4,5,6,7,8,9),3,3,byrow=TRUE)
> x
      [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
>
```

Macierze

```
> x<-matrix(c(1,2,3,4,5,6,7,8,9),3,3,byrow=TRUE)
> x
      [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
> x[1,]      ← pierwszy wiersz macierzy
[1] 1 2 3
> x[1:2,]    ← pierwsze dwa wiersze macierzy
      [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
>
```

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

61

Macierze

```
> x[,2]      ← druga kolumna
[1] 2 5 8
> x[,2:3]
      [,1] [,2]
[1,]    2    3
[2,]    5    6
[3,]    8    9
> x[,c(1,3)] ← pierwsza i trzecia kolumna
      [,1] [,2]
[1,]    1    3
[2,]    4    6
[3,]    7    9
>
```

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

62

Macierze

```
a<-matrix(c(2,4,6,3),2,2)

> a
      [,1] [,2]
[1,]    2    6
[2,]    4    3

> b<-matrix(c(4,3,2,7),2,2)

> b
      [,1] [,2]
[1,]    4    2
[2,]    3    7
```

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

63

Macierze

```
> a-b
      [,1] [,2]
[1,]   -2    4
[2,]    1   -4

> a*b
      [,1] [,2]
[1,]    8   12
[2,]   12   21

> a %*% b
      [,1] [,2]
[1,]   26   46
[2,]   25   29
```

← mnożenie odpowiadających sobie elementów

← mnożenie macierzy

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

64

Macierze

```

> b %% a
      [,1] [,2]
[1,]   16  30
[2,]   34  39
> t(a)
      [,1] [,2]
[1,]    2   4
[2,]    6   3

```

← mnożenie macierzy

← transponowanie macierzy

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

65

Macierze

```

> solve(a)
      [,1] [,2]
[1,] -0.1666667  0.3333333
[2,]  0.2222222 -0.1111111
> a %% solve(a)
      [,1] [,2]
[1,]    1   0
[2,]    0   1
>

```

← odwracanie macierzy

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

66

Rozwiązywanie układów równań liniowych

Układ równań:

$$3x_1 + x_2 + 4x_3 = 2$$

$$-x_1 + 6x_2 + x_3 = 1$$

$$x_1 + 2x_2 + 7x_3 = 2$$

```
> a<-matrix(c(3,1,4,-1,6,1,1,2,7),nrow=3,ncol=3,byrow=TRUE)
> a                                     ← macierz współczynników
      [,1] [,2] [,3]
[1,]    3    1    4
[2,]   -1    6    1
[3,]    1    2    7
> b<-c(2,1,2)                         ← macierz wyrazów wolnych
> solve(a,b)
[1] 0.3645833 0.1979167 0.1770833
>
```

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

67

Macierze

```
> a
      [,1] [,2] [,3]
[1,]    3    1    4
[2,]   -1    6    1
[3,]    1    2    7
> det(a)                               ← wyznacznik macierzy
[1] 96
>
```

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

68

Tworzenie macierzy poprzez łączenie wektorów

```

> x<-c(1,2,3,4,5)
> y<-c(6,7,8,9,10)
> z<-c(11,12,13,14,15)
> v<-rbind(x,y,z)      ← wektory stają się wierszami macierzy
> v
  [,1] [,2] [,3] [,4] [,5]
x    1    2    3    4    5
y    6    7    8    9   10
z   11   12   13   14   15
> v<-cbind(x,y,z)      ← wektory stają się kolumnami macierzy
> v
      x  y  z
[1,]  1  6 11
[2,]  2  7 12
[3,]  3  8 13
[4,]  4  9 14
[5,]  5 10 15
>

```

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

69

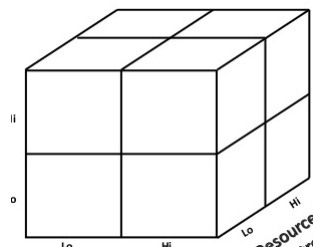
Tablice wielowymiarowe

array(data = NA, dim = length(data), dimnames = NULL)

```

> x<-array(c(1,2,3,4,5,6,7,8),dim=c(2,2,2))
> x
, , 1
      [,1] [,2]
[1,]    1    3
[2,]    2    4
, , 2
      [,1] [,2]
[1,]    5    7
[2,]    6    8
>

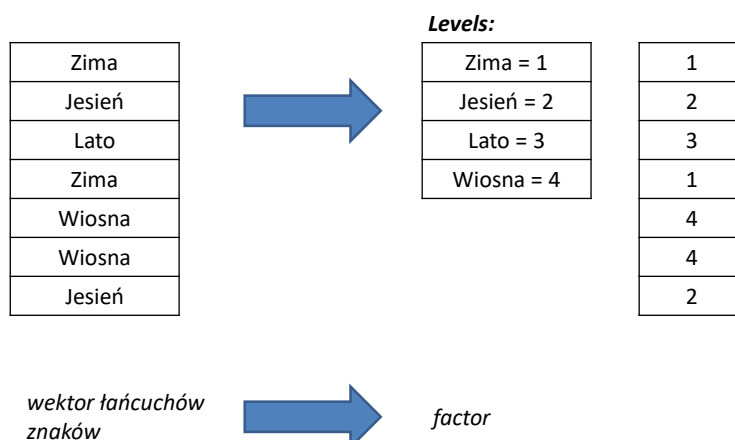
```



Paweł Lula, Katedra Systemów Obliczeniowych, UEK

70

Reprezentacja wartości nominalnych (Factors)



Paweł Lula, Katedra Systemów Obliczeniowych, UEK

71

Zmienne nominalne

```
> pory.roku<-c("zima","jesień","lato","zima","wiosna","wiosna","jesień")
> f<-factor(pory.roku)
> f
[1] zima  jesień lato   zima  wiosna wiosna jesień
Levels: jesień lato wiosna zima
> as.vector(f)
[1] "zima" "jesień" "lato" "zima" "wiosna" "wiosna" "jesień"
> table(f)
f
jesień  lato wiosna  zima
      2      1      2      2
>
```

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

72

Zmienne porządkowe

```
> pory.roku<-c("zima","jesień","lato","zima","wiosna","wiosna","jesień")

> f<-factor(pory.roku,levels=c("wiosna","lato","jesień","zima"),ordered = TRUE)

> f
[1] zima  jesień lato   zima  wiosna wiosna jesień
Levels: wiosna < lato < jesień < zima

> sort(f)
[1] wiosna wiosna lato   jesień jesień zima   zima
Levels: wiosna < lato < jesień < zima

> as.vector(f)
[1] "zima" "jesień" "lato" "zima" "wiosna" "wiosna" "jesień"

> table(f)
f
wiosna  lato jesień  zima
      2     1     2     2
>
```

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

73

Zamiana wartości numerycznych na nominalne

```
> p<-trunc(rnorm(100,mean=4000,sd=500))
> cut(p,breaks=c(-Inf,3500,4500,Inf),labels=c("Niska","Średnia","wysoka"))
[1] Średnia Średnia Średnia Średnia Średnia Niska Niska Średnia Niska Niska
[11] Średnia wysoka Średnia Średnia Średnia Średnia Średnia Średnia Średnia wysoka
[21] Średnia Średnia Średnia Średnia Średnia wysoka wysoka Średnia Średnia Średnia
[31] Średnia Średnia wysoka Średnia Średnia Średnia Średnia wysoka Średnia wysoka
[41] Niska Średnia Średnia Średnia Niska Średnia Średnia Średnia Średnia Średnia
[51] wysoka Średnia Niska Niska Średnia wysoka Średnia Średnia Średnia wysoka
[61] Średnia Średnia Średnia Średnia Średnia Niska Średnia Średnia Średnia Średnia
[71] Średnia Średnia Niska Średnia Średnia Niska Średnia Średnia Niska wysoka
[81] Średnia Średnia Średnia Średnia Średnia Średnia Średnia Średnia Średnia
[91] Niska Średnia Średnia Średnia Średnia Średnia Średnia Średnia Średnia
Levels: Niska Średnia wysoka
>
```

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

74

Zamiana wartości numerycznych na porządkowe

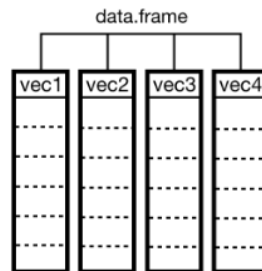
```
> p<-trunc(rnorm(100,mean=4000,sd=500))
> cut(p,breaks=c(-Inf,3500,4500,Inf),labels=c("Niska","Średnia","wysoka"),ordered=TRUE)
[1] Średnia Średnia Średnia Średnia Niska Średnia Średnia Średnia Średnia wysoka
[11] Średnia Średnia Średnia Średnia Średnia wysoka Średnia Średnia Średnia Niska
[21] wysoka Niska Średnia wysoka Niska Średnia Średnia Niska Niska Niska
[31] Średnia Niska Średnia wysoka Niska Średnia Średnia Średnia Niska Średnia
[41] Średnia Średnia Średnia Średnia Średnia wysoka Niska Niska wysoka Średnia
[51] Średnia Średnia Średnia Średnia Średnia Średnia Średnia Średnia wysoka Średnia
[61] Niska Średnia Średnia Średnia Średnia Niska wysoka Średnia Średnia Średnia
[71] Średnia wysoka Średnia Średnia Średnia Średnia Średnia Średnia Średnia Niska
[81] wysoka Średnia Niska Średnia Niska wysoka wysoka Średnia Średnia Średnia
[91] Średnia Średnia Średnia wysoka Średnia Średnia Średnia wysoka Średnia Średnia
Levels: Niska < Średnia < wysoka
>
```

Zamiana wartości porządkowych na numeryczne

```
> v<-cut(p,breaks=c(-Inf,3500,4500,Inf),labels=c("Niska","Średnia","wysoka"),ordered=TRUE)
> head(v)
[1] Średnia Średnia Średnia Średnia Niska Średnia
Levels: Niska < Średnia < wysoka
> as.integer(v)
[1] 2 2 2 2 1 2 2 2 2 3 2 2 2 2 3 2 2 2 1 3 1 2 3 1 2 2 1 1 1 2 1 2 3 1 2 2 2 1 2
[41] 2 2 2 2 2 3 1 1 3 2 2 2 2 2 2 2 2 3 2 1 2 2 2 2 1 3 2 2 2 3 2 2 2 2 2 2 1
[81] 3 2 1 2 1 3 3 2 2 2 2 2 3 2 2 2 3 2 2
> new.v<-c(150,300,450)
> new.v[v]
[1] 300 300 300 300 150 300 300 300 300 450 300 300 300 300 300 450 300 300 300 150
[21] 450 150 300 450 150 300 300 150 150 300 150 300 450 150 300 300 300 150 300
[41] 300 300 300 300 300 450 150 150 450 300 300 300 300 300 300 300 300 300 450 300
[61] 150 300 300 300 300 150 450 300 300 300 450 300 300 300 300 300 300 300 300 150
[81] 450 300 150 300 150 450 450 300 300 300 300 300 450 300 300 300 450 300 300 300
>
```

Ramki danych (obiekty typu data frame)

- Ramka danych (data frame) – struktura zbliżona do macierzy; złożona z kolumn (wektorów). Każdy z wektorów może przechowywać wartości różnych typów. Wartości przechowywane w jednym wektorze (kolumnie) muszą być tego samego typu.



Paweł Lula, Katedra Systemów Obliczeniowych, UEK

77

Ramki danych

Tworzenie ramki danych:

```
data.frame(vec1, vec2, ..., vecN, row.names = NULL, check.rows = FALSE, check.names = TRUE, stringsAsFactors = default.stringsAsFactors())
```

```
> a<-1:5
> b<-6:10
> c<-11:15
> df<-data.frame(a,b,c)
> df
  a  b  c
1 1  6 11
2 2  7 12
3 3  8 13
4 4  9 14
5 5 10 15
>
```

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

78

Ramki danych – definiowanie nazw wierszy

```
> nazwy<-paste("wiersz",1:5,sep="-")
> nazwy
[1] "wiersz-1" "wiersz-2" "wiersz-3" "wiersz-4" "wiersz-5"
> df<-data.frame(a,b,c,row.names=nazwy)
> df
```

	a	b	c
wiersz-1	1	6	11
wiersz-2	2	7	12
wiersz-3	3	8	13
wiersz-4	4	9	14
wiersz-5	5	10	15

```
>
```

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

79

Ramki danych – definiowanie nazw kolumn

```
> df<-data.frame(kol1=a,kol2=b,kol3=c)
> df
```

	kol1	kol2	kol3
1	1	6	11
2	2	7	12
3	3	8	13
4	4	9	14
5	5	10	15

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

80

Ramki danych – odwoływanie się do kolumn

```
> df<-data.frame(kol1=a,kol2=b,kol3=c)
> df
  kol1 kol2 kol3
1    1    6   11
2    2    7   12
3    3    8   13
4    4    9   14
5    5   10   15
> df$kol1
[1] 1 2 3 4 5
> df$kol3
[1] 11 12 13 14 15
```

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

81

Odwoływanie się do elementów ramki danych za pomocą indeksów

```
> df<-data.frame(kol1=a,kol2=b,kol3=c)
> df
  kol1 kol2 kol3
1    1    6   11
2    2    7   12
3    3    8   13
4    4    9   14
5    5   10   15
> df[1,1]
[1] 1
> df[1,2]
[1] 6
> df[,3]
[1] 11 12 13 14 15
> df[4,]
  kol1 kol2 kol3
4    4    9   14
>
```

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

82

Losowy wybór kolumn z ramki danych

```
> df<-data.frame(a,b,c)
> v<-sample(1:3,2,replace = FALSE)
> df.new<-df[,v]
> df.new
  a  c
1 1 11
2 2 12
3 3 13
4 4 14
5 5 15
>
```

← losowanie numerów dwóch kolumn

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

83

Wartości tekstowe w ramkach danych

```
> a
[1] 1 2 3 4 5
> b
[1] 6 7 8 9 10
> c
[1] 11 12 13 14 15
> d<-paste("s",1:5,sep="-")
> d
[1] "s-1" "s-2" "s-3" "s-4" "s-5"
> df<-data.frame(a,b,c,d)
> df
  a  b  c  d
1 1 6 11 s-1
2 2 7 12 s-2
3 3 8 13 s-3
4 4 9 14 s-4
5 5 10 15 s-5
> str(df)
'data.frame':   5 obs. of  4 variables:
 $ a: int  1 2 3 4 5
 $ b: int  6 7 8 9 10
 $ c: int 11 12 13 14 15
 $ d: Factor w/ 5 levels "s-1","s-2","s-3",...: 1 2 3 4 5
>
```

domyślnie wartości tekstowe są przekształcane na wartości typu factor

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

84

Wartości tekstowe w ramach danych

```
> df<-data.frame(a,b,c,d,stringsAsFactors=FALSE)
> df
  a  b  c  d
1 1  6 11 s-1
2 2  7 12 s-2
3 3  8 13 s-3
4 4  9 14 s-4
5 5 10 15 s-5
> str(df)
'data.frame':   5 obs. of  4 variables:
 $ a: int  1 2 3 4 5
 $ b: int  6 7 8 9 10
 $ c: int 11 12 13 14 15
 $ d: chr "s-1" "s-2" "s-3" "s-4" ...
>
```

Pozostawienie wartości tekstowych w obiekcie typu data frame

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

85

Wstawienie nowej kolumny do ramki danych

```
> df<-data.frame(a,b,c,d)
> df$e<-a*b
> df
  a  b  c  d  e
1 1  6 11 s-1  6
2 2  7 12 s-2 14
3 3  8 13 s-3 24
4 4  9 14 s-4 36
5 5 10 15 s-5 50
>
```

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

86

Usuwanie wybranych kolumn z ramki z danymi

*Zamierzamy usunąć
kolumnę „c” oraz „e”*

```
> str(df)
'data.frame':    5 obs. of  5 variables:
 $ a: int  1 2 3 4 5
 $ b: int  6 7 8 9 10
 $ c: int 11 12 13 14 15
 $ d: Factor w/ 5 levels "s-1","s-2","s-3",...: 1 2 3 4 5
 $ e: int  6 14 24 36 50
> names(df)          ← nazwy kolumn
[1] "a" "b" "c" "d" "e"
> is.element(names(df),c("c","e"))
[1] FALSE FALSE  TRUE FALSE  TRUE
> !is.element(names(df),c("c","e"))
[1]  TRUE  TRUE FALSE  TRUE FALSE
> df[,!is.element(names(df),c("c","e"))]
   a  b  d
1 1  6 s-1
2 2  7 s-2
3 3  8 s-3
4 4  9 s-4
5 5 10 s-5
> df<-df[,!is.element(names(df),c("c","e"))] ← zapamiętanie zmian!!!
>
```

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

87

Listy

Lista = uporządkowana sekwencja elementów
uporządkowana – dla każdego elementu (z wyjątkiem pierwszego i ostatniego) można wskazać element następny i poprzedni

Tworzenie listy

```
> ls<-list(5,4,3,2)
```

Dwie metody odwoływania się do elementów listy

```
> ls[3]          ← zwraca podlistę
[[1]]
[1] 3
```

```
> ls[[3]]        ← zwraca pojedynczy element
[1] 3
```

```
> ls[c(1,2)]
[[1]]
[1] 5
```

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

```
> ls[[c(1,2)]]
Error in ls[[c(1, 2)]] : subscript out of bounds
>
```

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

88

Listy

```

> list1<-list(1,2,3)
> list2<-list(4,5,6)
> vec1<-c(7,8,9)
> vec2<-c(10,11,12)
> c(vec1,vec2)           ← łączenie wektorów
[1] 7 8 9 10 11 12
> res1<-c(vec1,vec2)
> res1
[1] 7 8 9 10 11 12
> typeof(res1)          ← powstaje wektor
[1] "double"
>

```

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

89

Listy

```

> res2<-c(list1,list2)   ← łączenie list
> res2
[[1]]
[1] 1

[[2]]
[1] 2

[[3]]
[1] 3

[[4]]
[1] 4

[[5]]
[1] 5

[[6]]
[1] 6

> typeof(res2)           ← powstaje lista
[1] "list"
>

```

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

90

Listy

```

> res3<-c(list1,vec1)      <- łączenie listy i wektora
> res3
[[1]]
[1] 1

[[2]]
[1] 2

[[3]]
[1] 3

[[4]]
[1] 7

[[5]]
[1] 8

[[6]]
[1] 9

> typeof(res3)             <- powstaje lista
[1] "list"
>

```

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

91

Listy

```

> res4<-append(list1,vec1) <- łączenie listy i wektora
> res4
[[1]]
[1] 1

[[2]]
[1] 2

[[3]]
[1] 3

[[4]]
[1] 7

[[5]]
[1] 8

[[6]]
[1] 9

> typeof(res4)             <- powstaje lista
[1] "list"
>

```

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

92

Listy

```
> res5<-append(list1,vec1,after=2)  ← wstawianie elementów do listy
> res5
[[1]]
[1] 1

[[2]]
[1] 2

[[3]]
[1] 7

[[4]]
[1] 8

[[5]]
[1] 9

[[6]]
[1] 3

>
```

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

93

Usuwanie elementów z listy

```
> lista<-list(5,3,7,5,6,2,3,4,5,9)
> length(lista)
[1] 10
> lista<-lista[-c(2,3)]  ← usuwanie drugiego i trzeciego elementu listy
> length(lista)
[1] 8
```

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

94

Lista list

```
> res6<-list(list1,list2)      ← tworzenie listy list
> res6
[[1]]
[[1]][[1]]
[1] 1

[[1]][[2]]
[1] 2

[[1]][[3]]
[1] 3

[[2]]
[[2]][[1]]
[1] 4

[[2]][[2]]
[1] 5

[[2]][[3]]
[1] 6

>
```

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

95

FUNKCJE W JĘZYKU R

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

96

Funkcje – definiowanie i wywoływanie

Definiowanie funkcji:

```
obiekt <- function(parametry-formalne) definicja-funkcji
```

Wywołanie funkcji:

```
obiekt(parametry-aktualne)
```

Wyświetlanie definicji funkcji:

```
obiekt
```

Przykład:

```
> multiplyByTwo<-function(x) 2*x
> multiplyByTwo(44)
[1] 88
> multiplyByTwo
function(x) 2*x
>
```

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

97

Dwie postacie funkcji

a) one-line:

```
obiekt <- function (parametry) wyrażenie
```

b) multi-line:

```
obiekt <- function(parametry) {
  definicja funkcji (jeden lub więcej wierszy)
}
```

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

98

Definiowanie wartości zwracanej przez funkcję

a) zwracanie wartości ostatniego wyrażenia

```
> f <- function(a, b) {
+   min(a,b)
+ }
> f(4,2)
[1] 2
```

b) poprzez użycie instrukcji „return”

```
> test<-function(x) {
+   print("step 1")
+   return (2*x)
+   print("step 2")
+ }
> test(5)
[1] "step 1"
[1] 10
>
```

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

99

Kolejność parametrów formalnych i aktualnych

a) kolejność parametrów formalnych i aktualnych musi być taka sama

```
> s <- function(a,b) 3*a+b
> s(2,3)
[1] 9
```

b) możliwość zmiany kolejności parametrów

```
> s(a=2,b=3)
[1] 9
> s(b=2,a=3)
[1] 11
>
```

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

100

Domyślne wartości parametrów

```
> fun <- function(a=1,b) a * b
> fun(3,4)
[1] 12
> fun(3)
Error in fun(3) : argument "b" is missing, with no default
> fun(b=3)
[1] 3
>
```

Zwykle parametry mające określoną wartość domyślną definiowane są na końcu listy parametrów

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

101

Parametry opcjonalne

Opcjonalne = nieobowiązkowe przy wywołaniu funkcji

```
> s <- function(...) {
+ sum(...)
+ }
> s(3,4,5,5,5,9)
[1] 31
```

Parametry opcjonalne muszą znajdować się na końcu listy parametrów

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

102

Funkcja jako parametr

Funkcja może zostać przekazana do innej funkcji jako parametr

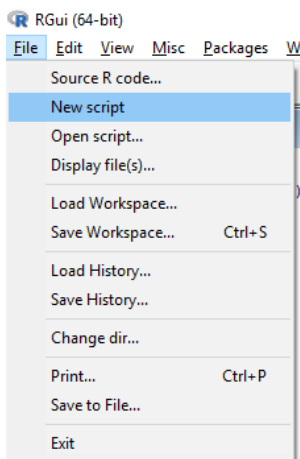
wywołanie funkcji przekazanej jako parametr:

`do.call(funkcja,lista-parametrów-aktualnych)`

```
> p <- function(x) 2*x
> f <- function(x,y) do.call(x,y)
> f(p,50)
Error in do.call(x, y) : second argument must be a list
> f(p,list(50))
[1] 100
```

SKRYPTY

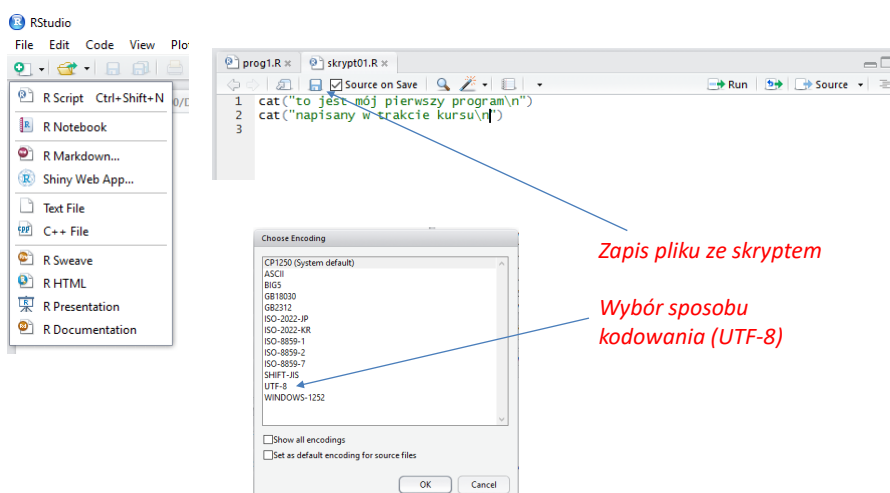
Tworzenie skryptów – środowisko R



Paweł Lula, Katedra Systemów Obliczeniowych, UEK

105

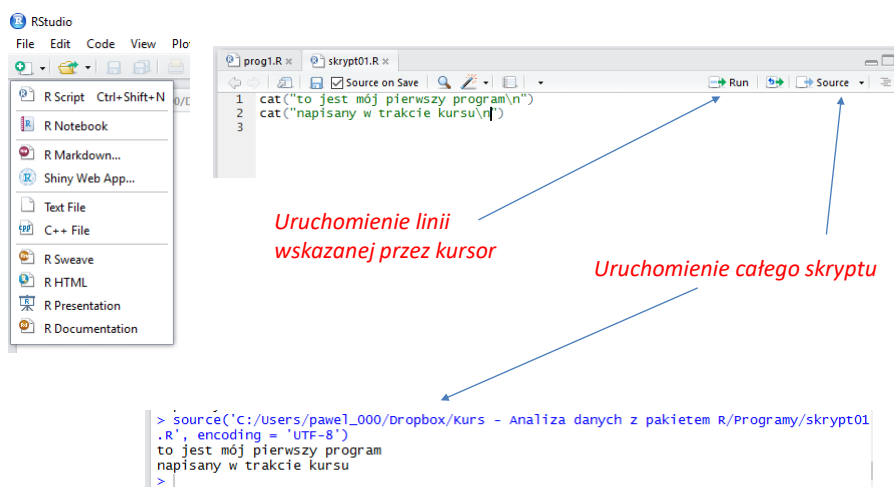
Tworzenie skryptów – środowisko RStudio



Paweł Lula, Katedra Systemów Obliczeniowych, UEK

106

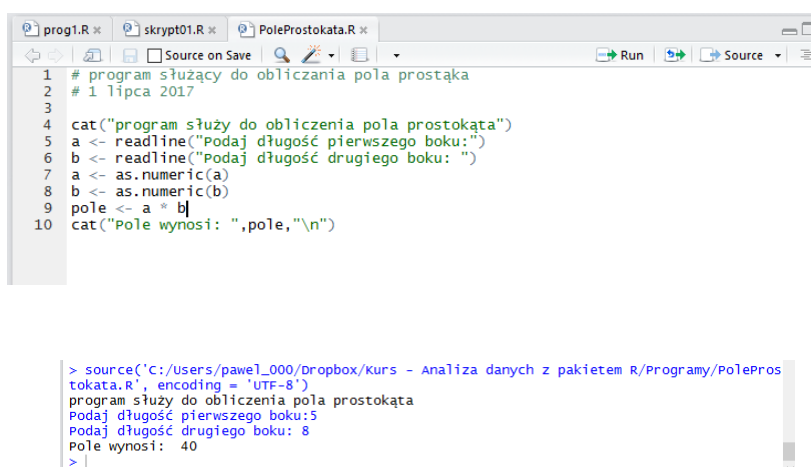
Tworzenie skryptów – środowisko RStudio



Paweł Lula, Katedra Systemów Obliczeniowych, UEK

107

Drugi program...



Paweł Lula, Katedra Systemów Obliczeniowych, UEK

108

INSTRUKCJE STERUJĄCE JĘZYKA R

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

109

Instrukcje sterujące

- if
- switch
- pętle:
 - for
 - while
 - repeat
- break
- next

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

110

Instrukcja if

Instrukcja if:

```
if (warunek) instrukcja-1 else instrukcja-2
```

lub:

```
if (warunek) {
    sekwencja-instrukcji-1
} else {
    sekwencja-instrukcji-2
}
```

Instrukcja if

Przykład:

```
> x<-5
> if (x==5) cat("yes\n") else cat("no\n")
yes
> if (x==5) {
+ cat("YES! ")
+ cat("x is equal to 5\n")
+ }
YES! x is equal to 5
```

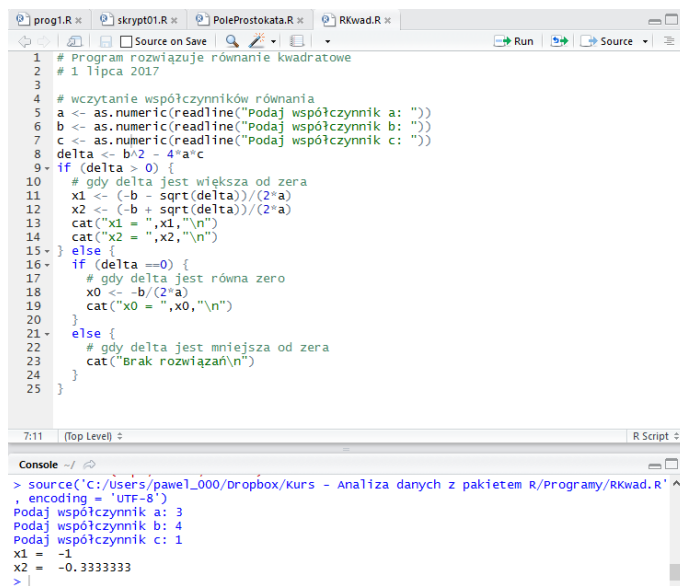

Użycie instrukcji if w instrukcji przypisania

```
> x <- 6
> y <- if (x%%2==0) 0 else 1
> y
[1] 0
> y <- if (x%%2==0) "even number" else "odd number"
> y
[1] "even number"
>
```

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

113

Równanie kwadratowe



```
1 # Program rozwiązuje równanie kwadratowe
2 # 1 lipca 2017
3
4 # wczytanie współczynników równania
5 a <- as.numeric(readline("Podaj współczynnik a: "))
6 b <- as.numeric(readline("Podaj współczynnik b: "))
7 c <- as.numeric(readline("Podaj współczynnik c: "))
8 delta <- b^2 - 4*a*c
9
10 if (delta > 0) {
11   # gdy delta jest większa od zera
12   x1 <- (-b - sqrt(delta))/(2*a)
13   x2 <- (-b + sqrt(delta))/(2*a)
14   cat("x1 = ", x1, "\n")
15   cat("x2 = ", x2, "\n")
16 }
17 else {
18   if (delta == 0) {
19     # gdy delta jest równa zero
20     x0 <- -b/(2*a)
21     cat("x0 = ", x0, "\n")
22   }
23   else {
24     # gdy delta jest mniejsza od zera
25     cat("Brak rozwiązań\n")
26   }
27 }
```

```
> source('c:/Users/pawel_000/Dropbox/Kurs - Analiza danych z pakietem R/Programy/Rkwad.R')
, encoding = 'UTF-8')
Podaj współczynnik a: 3
Podaj współczynnik b: 4
Podaj współczynnik c: 1
x1 = -1
x2 = -0.3333333
>
```

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

114

Równanie kwadratowe – użyciem funkcji

```

1 # Program rozwiązuje równanie kwadratowe
2 # wersja z zastosowaniem funkcji
3 # 1 lipca 2017
4
5 # wczytanie <- function() {
6 #   # wczytanie współczynników równania
7   a <- as.numeric(readline("Podaj współczynnik a: "))
8   b <- as.numeric(readline("Podaj współczynnik b: "))
9   c <- as.numeric(readline("Podaj współczynnik c: "))
10  return (c(a,b,c))
11 }
12
13 delta <- function(wsp) {
14  return(wsp[2]^2 - 4*wsp[1]*wsp[3])
15 }
16
17 # rozwiązanie <- function(wsp,delta) {
18  if (delta > 0) {
19    # gdy delta jest większa od zera
20    x1 <- (-wsp[2] + sqrt(delta))/(2*wsp[1])
21    x2 <- (-wsp[2] - sqrt(delta))/(2*wsp[1])
22    return (list(x1,x2))
23  } else {
24    if (delta == 0) {
25      # gdy delta jest równa zero
26      x0 <- -wsp[2]/(2*wsp[1])
27      return(list(x0))
28    } else {
29      # gdy delta jest mniejsza od zera
30      return(list())
31    }
32  }
33 }
34
35 # wyświetlenie <- function(res){
36  if (length(res) == 2) {
37    cat("x1 = ",res[[1]],"\n")
38    cat("x2 = ",res[[2]],"\n")
39  } else {
40    if (length(res) == 1) {
41      cat("x0 = ",res[[1]],"\n")
42    } else {
43      cat("brak rozwiązań")
44    }
45  }
46 }
47
48 cat("Program służy do rozwiązywania równania kwadratowego")
49 wsp <- wczytanie()
50 d <- delta(wsp)
51 list <- rozwiązanie(wsp,d)
52 wyświetlenie(list)
53
54

```

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

115

Równanie kwadratowe – użyciem funkcji

```

1 # Program rozwiązuje równanie kwadratowe
2 # wersja z zastosowaniem funkcji
3 # 1 lipca 2017
4
5 # wczytanie <- function() {
6 #   # wczytanie współczynników równania
7   a <- as.numeric(readline("Podaj współczynnik a: "))
8   b <- as.numeric(readline("Podaj współczynnik b: "))
9   c <- as.numeric(readline("Podaj współczynnik c: "))
10  return (c(a,b,c))
11 }
12
13 delta <- function(wsp) {
14  return(wsp[2]^2 - 4*wsp[1]*wsp[3])
15 }
16
17 # rozwiązanie <- function(wsp,delta) {
18  if (delta > 0) {
19    # gdy delta jest większa od zera
20    x1 <- (-wsp[2] + sqrt(delta))/(2*wsp[1])
21    x2 <- (-wsp[2] - sqrt(delta))/(2*wsp[1])
22    return (list(x1,x2))
23  } else {
24    if (delta == 0) {
25      # gdy delta jest równa zero
26      x0 <- -wsp[2]/(2*wsp[1])
27      return(list(x0))
28    } else {
29      # gdy delta jest mniejsza od zera
30      return(list())
31    }
32  }
33 }
34
35 # wyświetlenie <- function(res){
36  if (length(res) == 2) {
37    cat("x1 = ",res[[1]],"\n")
38    cat("x2 = ",res[[2]],"\n")
39  } else {
40    if (length(res) == 1) {
41      cat("x0 = ",res[[1]],"\n")
42    } else {
43      cat("brak rozwiązań")
44    }
45  }
46 }
47
48 cat("Program służy do rozwiązywania równania kwadratowego")
49 wsp <- wczytanie()
50 d <- delta(wsp)
51 list <- rozwiązanie(wsp,d)
52 wyświetlenie(list)
53
54

```

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

116

Równanie kwadratowe – użyciem funkcji

```

1 # Program służący do rozwiązywania równania kwadratowego
2 # a*x^2 + b*x + c = 0
3 # a, b, c - współczynniki
4 # x1, x2 - rozwiązania
5 # x0 - rozwiązanie
6 # x - lista rozwiązań
7 # x - lista rozwiązań
8 # x - lista rozwiązań
9 # x - lista rozwiązań
10 # x - lista rozwiązań
11 # x - lista rozwiązań
12 # x - lista rozwiązań
13 # x - lista rozwiązań
14 # x - lista rozwiązań
15 # x - lista rozwiązań
16 # x - lista rozwiązań
17 # x - lista rozwiązań
18 # x - lista rozwiązań
19 # x - lista rozwiązań
20 # x - lista rozwiązań
21 # x - lista rozwiązań
22 # x - lista rozwiązań
23 # x - lista rozwiązań
24 # x - lista rozwiązań
25 # x - lista rozwiązań
26 # x - lista rozwiązań
27 # x - lista rozwiązań
28 # x - lista rozwiązań
29 # x - lista rozwiązań
30 # x - lista rozwiązań
31 # x - lista rozwiązań
32 # x - lista rozwiązań
33 # x - lista rozwiązań
34 # x - lista rozwiązań
35 # x - lista rozwiązań

```

```

16
17 - rozwiązanie <- function(wsp,delta) {
18 -   if (delta > 0) {
19     # gdy delta jest większa od zera
20     x1 <- (-wsp[2] - sqrt(delta))/(2*wsp[1])
21     x2 <- (-wsp[2] + sqrt(delta))/(2*wsp[1])
22     return (list(x1,x2))
23   } else {
24     if (delta == 0) {
25       # gdy delta jest równa zero
26       x0 <- -wsp[2]/(2*wsp[1])
27       return(list(x0))
28     } else {
29       # gdy delta jest mniejsza od zera
30       return(list())
31     }
32   }
33 }
34 }
35

```

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

117

Równanie kwadratowe – użyciem funkcji

```

1 # Program służący do rozwiązywania równania kwadratowego
2 # a*x^2 + b*x + c = 0
3 # a, b, c - współczynniki
4 # x1, x2 - rozwiązania
5 # x0 - rozwiązanie
6 # x - lista rozwiązań
7 # x - lista rozwiązań
8 # x - lista rozwiązań
9 # x - lista rozwiązań
10 # x - lista rozwiązań
11 # x - lista rozwiązań
12 # x - lista rozwiązań
13 # x - lista rozwiązań
14 # x - lista rozwiązań
15 # x - lista rozwiązań
16 # x - lista rozwiązań
17 # x - lista rozwiązań
18 # x - lista rozwiązań
19 # x - lista rozwiązań
20 # x - lista rozwiązań
21 # x - lista rozwiązań
22 # x - lista rozwiązań
23 # x - lista rozwiązań
24 # x - lista rozwiązań
25 # x - lista rozwiązań
26 # x - lista rozwiązań
27 # x - lista rozwiązań
28 # x - lista rozwiązań
29 # x - lista rozwiązań
30 # x - lista rozwiązań
31 # x - lista rozwiązań
32 # x - lista rozwiązań
33 # x - lista rozwiązań
34 # x - lista rozwiązań
35 # x - lista rozwiązań

```

```

36 - wyswietlenie <- function(res){
37 -   if (length(res) == 2) {
38     cat("x1 = ",res[[1]],"\n")
39     cat("x2 = ",res[[2]],"\n")
40   } else {
41     if (length(res) == 1) {
42       cat("x0 = ",res[[1]],"\n")
43     } else {
44       cat("Brak rozwiązań")
45     }
46   }
47 }
48
49 cat("Program służący do rozwiązywania równania kwadratowego")
50 wsp <- wczytanie()
51 d <- delta(wsp)
52 lst <- rozwiązanie(wsp,d)
53 wyswietlenie(lst)
54

```

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

118

Instrukcja switch

Instrukcja switch:
`switch (wyrażenie, lista)`

Przykłady:

```
> switch(2, "red", "green", "blue")
[1] "green"

> switch(1, "red", "green", "blue")
[1] "red"

> switch(3, "a", "b", "c", "d", "e")
[1] "c"

> switch(2, 5, 6, 7, 8, 9)
[1] 6

> switch(8, 5, 6, 7, 8, 9)
NULL
```

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

119

Równanie kwadratowe po raz trzeci...



```
1 # Program rozwiązuje równanie kwadratowe
2 # 1 lipca 2017
3
4 # wczytanie współczynników równania
5 a <- as.numeric(readline("Podaj współczynnik a: "))
6 b <- as.numeric(readline("Podaj współczynnik b: "))
7 c <- as.numeric(readline("Podaj współczynnik c: "))
8 delta <- b^2 - 4*a*c
9 x <- switch(sign(delta)+2,
10            "Brak rozwiązań",
11            -b/(2*a),
12            c((-b-sqrt(delta))/(2*a), (-b+sqrt(delta))/(2*a)))
13 print(x)
14
```

```
> source('C:/Users/pawel_000/Dropbox/Kurs - Analiza danych z pakietem R/Programy/RKwad2.R', encoding = 'UTF-8')
Podaj współczynnik a: 1
Podaj współczynnik b: 1
Podaj współczynnik c: 1
[1] "Brak rozwiązań"
>
```

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

120

Instrukcja for

Instrukcja for:

```
for (zmienna in lista-lub-wektor) instrukcja
```

lub:

```
for (zmienna in lista-lub-wektor) {
    sekwencja-instrukcji
}
```

Instrukcja for

```
for (zmienna in lista-lub-wektor) {
    instrukcja-1
    instrukcja-2
    instrukcja-3
    ...
    instrukcja-N
}
```

ten fragment jest wielokrotnie powtarzany

ile razy?

Instrukcja for

```
for (zmienna in lista-lub-wektor) {
    instrukcja-1
    instrukcja-2
    instrukcja-3
    ...
    instrukcja-N
}
```

ten fragment jest wielokrotnie powtarzany

ile razy?

liczba powtórzeń pętli jest równa liczbie elementów w wektorze (lub liście)

--	--	--	--	--	--	--	--

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

123

Instrukcja for

```
for (zmienna in lista-lub-wektor) {
    instrukcja-1
    instrukcja-2
    instrukcja-3
    ...
    instrukcja-N
}
```

ten fragment jest wielokrotnie powtarzany

przy kolejnych powtórzeniach pętli wartość zmiennej jest równa wartości przechowywanej na kolejnej pozycji wektora (lub listy)

ile razy?

liczba powtórzeń pętli jest równa liczbie elementów w wektorze (lub liście)

4	2	3	7	5	5	1
---	---	---	---	---	---	---

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

124

Pierwsza pętla for...

```
PetlaFor01.R
1 # przykładowe zastosowanie pętli for
2
3 for (i in 1:5) {
4   cat("Zmienna i ma wartość: ")
5   cat(i)
6   cat("\n")
7 }
```

```
> source('C:/users/pawel_000/Dropbox/
01.R', encoding = 'UTF-8')
Zmienna i ma wartość: 1
Zmienna i ma wartość: 2
Zmienna i ma wartość: 3
Zmienna i ma wartość: 4
Zmienna i ma wartość: 5
> |
```

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

125

Druga pętla for...

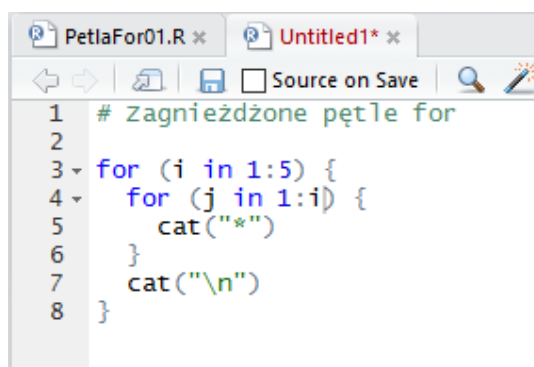
```
PetlaFor01.R
1 # przykładowe zastosowanie pętli for
2
3 pory.roku <- c("wiosna", "lato", "jesień", "zima")
4 for (i in pory.roku) {
5   cat("Zmienna i ma wartość: ")
6   cat(i)
7   cat("\n")
8 }
```

```
> source('C:/users/pawel_000/Dropbox/
01.R', encoding = 'UTF-8')
Zmienna i ma wartość: wiosna
Zmienna i ma wartość: lato
Zmienna i ma wartość: jesień
Zmienna i ma wartość: zima
> |
```

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

126

Dwie pętle...



```

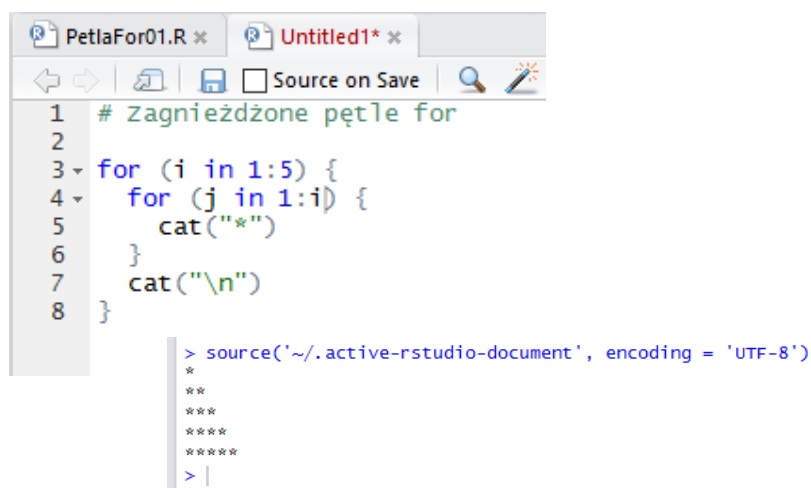
1 # zagnieżdżone pętle for
2
3 for (i in 1:5) {
4   for (j in 1:i) {
5     cat("*")
6   }
7   cat("\n")
8 }

```

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

127

Dwie pętle...



```

1 # zagnieżdżone pętle for
2
3 for (i in 1:5) {
4   for (j in 1:i) {
5     cat("*")
6   }
7   cat("\n")
8 }

```

```

> source('~/.active-rstudio-document', encoding = 'UTF-8')
*
*
*
*
*
*
*
*
*
*
>

```

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

128

Sumowanie liczb całkowitych

```
PetiaFor01.R x Untitled1* x Silnia.R* x Suma.R* x
Source on Save Run
1 # obliczenie sumy liczb: 1 + 2 + ... + N
2
3 n <- readline("Podaj n: ")
4 suma <- 0
5
6 for (i in 1:n) suma <- suma + i
7
8 cat("Suma liczb całkowitych od 1 do ",n," wynosi: ", suma,"\n")
9 |
```

```
> source('C:/Users/pawel_000/Dropbox/kurs - Analiz
encoding = 'UTF-8')
Podaj n: 5
Suma liczb całkowitych od 1 do 5 wynosi: 15
> |
```

Silnia

```
PetiaFor01.R x Untitled1* x Silnia.R* x Suma.R* x
Source on Save
1 # obliczenie wartości n!
2
3 n <- readline("Podaj n: ")
4
5 silnia <- 1
6
7 for (i in 1:n) silnia <- silnia * i
8
9 cat(n,"! =",silnia,"\n")|
```

```
> source('C:/Users/pawel_000
', encoding = 'UTF-8')
Podaj n: 5
5 ! = 120
> |
```

Instrukcja while

Instrukcja while:

```
while (warunek) instrukcja
```

lub:

```
while (warunek) {  
    sekwencja-instrukcji  
}
```

Instrukcja while

```
while (warunek) {  
    instrukcja-1  
    instrukcja-2  
    instrukcja-3  
    ...  
    instrukcja-N  
}
```

ten fragment jest wielokrotnie powtarzany

ile razy?

Instrukcja while

```
while (warunek) {
    instrukcja-1
    instrukcja-2
    instrukcja-3
    ...
    instrukcja-N
}
```

ten fragment jest wielokrotnie powtarzany

ile razy?

tak długo, jak długo spełniony jest warunek logiczny

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

133

Instrukcja while

```
while (warunek) {
    instrukcja-1
    instrukcja-2
    instrukcja-3
    ...
    instrukcja-N
}
```

ten fragment jest wielokrotnie powtarzany

ile razy?

tak długo, jak długo spełniony jest warunek logiczny

UWAGA:
realizacja instrukcji wchodzących w skład pętli musi doprowadzić do zmiany wartości warunku logicznego!
W przeciwnym przypadku pętla nigdy nie zakończy swojego działania.

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

134

Silnia po raz drugi

```

1 # obliczenie wartości n!
2
3 n <- as.numeric(readline("Podaj n: "))
4 i <- n
5
6 silnia <- 1
7
8 while (i > 0) {
9   silnia <- silnia * i
10  i <- i - 1
11 }
12
13 cat(n, "! =", silnia, "\n")

```

```

> source('C:/Users/pawe'
', encoding = 'UTF-8')
Podaj n: 5
5 ! = 120
> |

```

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

135

Instrukcja repeat

Instrukcja repeat:

```
repeat instrukcja
```

lub:

```
repeat {
  sekwencja-instrukcji
}
```

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

136

Instrukcja repeat

```
repeat {
    instrukcja-1
    instrukcja-2
    instrukcja-3
    ...
    instrukcja-N
}
```

ten fragment jest wielokrotnie powtarzany

ile razy?

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

137

Instrukcja repeat

```
repeat {
    instrukcja-1
    instrukcja-2
    instrukcja-3
    ...
    instrukcja-N
}
```

ten fragment jest wielokrotnie powtarzany

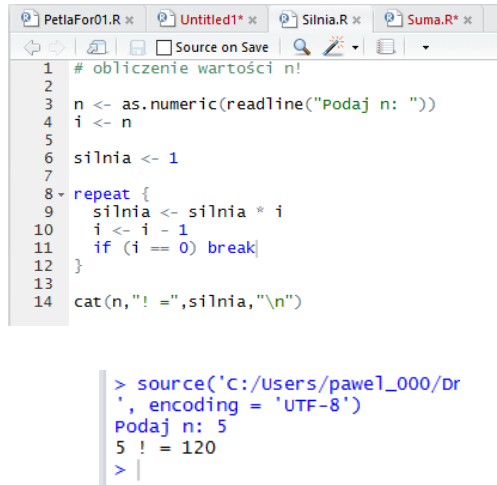
ile razy?

aż do momentu wykonania instrukcji break przerywającej działanie pętli

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

138

Silnia po raz trzeci



```

1 # obliczenie wartości n!
2
3 n <- as.numeric(readline("Podaj n: "))
4 i <- n
5
6 silnia <- 1
7
8 repeat {
9   silnia <- silnia * i
10  i <- i - 1
11  if (i == 0) break
12 }
13
14 cat(n, "! =", silnia, "\n")

```

```

> source('C:/users/pawel_000/dr
', encoding = 'UTF-8')
Podaj n: 5
5 ! = 120
> |

```

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

139

Instrukcje break i next

- break – kończy wykonywanie pętli (repeat, while, for)
- next – kończy bieżące powtórzenie pętli i przechodzi do kolejnego

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

140

Suma liczb parzystych – dwa rozwiązania problemu...

```

1 # wyznaczenie sumy liczb parzystych z przedziału od 1 do N
2
3 n <- as.numeric(readline("Podaj n: "))
4
5 suma <- 0
6
7 for (i in 1:n) {
8   if (i%%2 == 0) suma <- suma + i
9 }
10
11 cat("Suma wynosi: ",suma,"\n")

> source('C:/Users/paw
.R', encoding = 'UTF-8
Podaj n: 100
Suma wynosi: 2550
>

```

```

1 # wyznaczenie sumy liczb parzystych z przedziału od 1 do N
2
3 n <- as.numeric(readline("Podaj n: "))
4
5 suma <- 0
6
7 for (i in 1:n) {
8   if (i%%2 != 0) next
9   suma <- suma + i
10 }
11
12 cat("Suma wynosi: ",suma,"\n")

```

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

141

Paweł Lula, Katedra Systemów Obliczeniowych,
 Uniwersytet Ekonomiczny w Krakowie
 pawel.lula@uek.krakow.pl

DZIĘKUJĘ ZA UWAGĘ!

Paweł Lula, Katedra Systemów Obliczeniowych, UEK

142