Wprowadzenie do języka R

Wojciech Jaworski
Instytut Informatyki
Uniwersytet Warszawski

1 Podstawy

strona programu: `http://www.r-project.org/`

wprowadzenie: `http://cran.r-project.org/doc/manuals/R-intro.html`
opis jerzyka: `http://cran.r-project.org/doc/manuals/R-lang.html`

uruchamianie:
`R`

kończenie:
```
> q()
```

wyrażenia matematyczne:
```
> 1+1
[1] 2
> sqrt(30)
[1] 5.477226
> log(100)
[1] 4.60517
> log(100,10)
[1] 2
> pi
[1] 3.141593
> sin(30*pi/180)
[1] 0.5
> 1/(2*sqrt(45*89))
[1] 0.007900758
> 16^(0.5)
[1] 4
```

modulo
```
> 123 %% 10
[1] 3
```

dzielenie całkowite

```
> 123 %/% 10
[1] 12
```

dokańczanie wyrażeń:

```
> 2*
+ 2
[1] 4
```

komentarze od # do końca linii

definiowanie stałych:

```
> x <- 2
> x
[1] 2
```

definiowanie funkcji, aplikacja:

```
> kwadrat <- function(x) x*x
> kwadrat(3)
> kwadrat(x=3)
```

każdy argument funkcji możemy wywołać przez podanie nazwy.

argumenty domyślne:

```
> kwadrat <- function(x=5) x*x
> kwadrat(3)
> kwadrat()
```

operatory logiczne:

`<, <=, >, >=, ==, !=`
`&, |, !` - wektorowe
`&&, ||` - leniwe

instrukcja warunkowa:

```
> rowne <- function(x,y) if (x == y) "a" else 'b'
> rowne(1,1)
[1] "a"
> rowne(1,2)
[1] "b"
```

rekurencja:

```
> mnoz <- function(x,y,w,fun)
+ if(y==0) w else fun(x,y-1,w+x,fun)
> mnoz(4,5,0,mnoz)
```

Zadanie 1

Napisz procedurę obliczającą silnię funkcyjnie (za pomocą rekurencji) i policz 5!

Zadanie 2 *dodatkowe, niepunktowane*

Napisz procedurę, która przekształca daną liczbę w taką, w której cyfry występują w odwrotnej kolejności, np. 1234 jest przekształcane na 4321.

2 Imperatywność

w R mamy tylko wyrażenia, nie ma instrukcji
grupowanie wyrażeń: `{expr_1; ...; expr_m}`
koniec linii jest równoważny średnikowi

wykonanie operacji dla każdego obiektu ze zbioru:
`for (name in expr_1) expr_2`

```
> x<-0
> for(i in 1:20) x <- x+i
> x
[1] 210
```

powtarzanie w nieskończoność:
`repeat expr`

pętla z warunkiem:
`while (condition) expr`

przerwanie pętli:
`break`

Zadanie 3

Napisz procedurę obliczającą silnię imperatywnie i policz 5!

wczytanie pliku z poleceniami (nie wypisuje wyników działania komend):

```
> source("commands.R")
```

wypisanie wartości na stdout:
`print(x)`

przekierowanie wyjścia interpretera do pliku:

```
> sink("record.lis")
```

przekierowanie wyjścia interpretera na ekran:

```
> sink()
```

wypisanie nazw obiektów zawartych w pamięci:

```
> objects()
```

usunięcie obiektów z pamięci:

```
> rm(x, y, z, ink, junk, temp, foo, bar)
```

zapisywanie sesji przy wyjściu, pliki:

```
.RData
.Rhistory
```

wczytywane automatycznie przy uruchamianiu

3 Wektory

liczba jest jednoelementowym wektorem

generowanie:

```
> x <- c(10.4, 5.6, 3.1, 6.4, 21.7)
> x
[1] 10.4  5.6  3.1  6.4 21.7
```

automatyczny flatten:

```
> y <- c(x, 0, x)
> y
[1] 10.4  5.6  3.1  6.4 21.7  0.0 10.4  5.6  3.1  6.4 21.7
```

sekwencje kolejnych liczb:

```
> 1:9
[1] 1 2 3 4 5 6 7 8 9
> 30:1
> seq(-5,5,by=0.2)
> seq(-5,5,length=51)
> rep(c(1,2),2)
[1] 1 2 1 2
```

`help` pozwala dowiedzieć się jakie są opcje przy `seq`:

```
> help()
     help(topic, package = NULL, lib.loc = NULL,
          verbose = getOption("verbose"),
          try.all.packages = getOption("help.try.all.packages"),
          help_type = getOption("help_type"))

> help(seq)
```

nazwy atrybutów można skracać:

```
> seq(-5,5,length=51)
> seq(-5,5,lengt=51)
> seq(-5,5,leng=51)
```

automatyczny map:

```
> x+1
[1] 11.4  6.6  4.1  7.4 22.7
> 1/x
[1] 0.09615385 0.17857143 0.32258065 0.15625000 0.04608295
> x > 5
[1]  TRUE  TRUE FALSE  TRUE  TRUE
```

`ifelse(condition, a, b)` - if działający na wektorach
`&` i `|` - operatory logiczne działające na wektorach
przykład

```
> ifelse(x>5,x+1,x-1)
```

branie podciągu (w tym elementu pod danym indeksem):

```
> x[3]
[1] 3.1
> x[10]
[1] NA
```

```
> x[c(3,4)]
[1] 3.1 6.4
> x[-3]
[1] 10.4  5.6  6.4 21.7
> x[c(TRUE,FALSE,TRUE)]
[1] 10.4  3.1  6.4
```
maska została wydłużona przez jej powtórzenie
```
> x[x > 5]
[1] 10.4  5.6  6.4 21.7
```
funkcje agregujące:
`max, min, mean, median, length, sum, prod, sort, diff, rank, which.max`

brakujące wartości:
specjalna wartość `NA`
`is.na(x)` zwraca wektor wartości logicznych o tej samej długości, co `x` z wartościami `TRUE` na polach na których `x` ma wartość `NA`
`NaN` — Not a Number — wynik wykonania niezdefiniowanej operacji arytmetycznej np $0/0$.
`is.na(x)` zwraca `TRUE` zarówno dla `NA` jak i `NaN`
`is.nan(x)` zwraca `TRUE` jedynie dla `NaN`

Zadanie 4

Stwórz wektor x taki,że $x_i = \sin(i)$, $i \leq 100$. Zdyskretyzuj go, czyli utwórz wektor wartości nominalnych y taki, że

$$y_i = \begin{cases} \text{A} & \text{gdy } x_i < -0.5, \\ \text{B} & \text{gdy } x_i \in \langle -0.5, 0.5 \rangle, \\ \text{C} & \text{wpp.} \end{cases}$$

Zadanie należy rozwiązać nie korzystając z pętli ani rekurencji.

wektory napisów
`paste(c("X","Y"), 1:10, sep="")`

Wektor jest traktowany w R jako atomowa struktura danych (nie ma zwykłych liczb). Wektor zawiera komponenty które mają swój typ: `numeric`, `logical`, `character`, ...

informacja o typie:
```
> mode(x)
```
zmiana typu:
```
> mode(x) <- "character"
```
`mode` jest atrybutem struktury danych, można go zmieniać.!
```
> as.character(x)
> is.character(x)
> as.integer(x)
> as.logical(x)
```
kropka nie jest symbolem operatorem.

tworzenie wektora ustalonej długości:

```
> numeric(5)
> character(0)
```

długość wektora i jej zmiana:

```
> length(x)
> length(x) <- 3
> x
[1] "10.4" "5.6"  "3.1"
> length(x) <- 5
> x
[1] "10.4" "5.6"  "3.1"  NA     NA
```

pusty wektor

```
> e <- numeric()
> e
numeric(0)
> e[3] <- 17
> e
[1] NA NA 17
```

wektor z nazwanymi współrzędnymi i atrybut `names`:

```
> fruit <- c(5, 10, 1, 20)
> names(fruit) <- c("orange", "banana", "apple", "peach")
> lunch <- fruit[c("apple","orange")]
> fruit
> fruit["apple"]
```