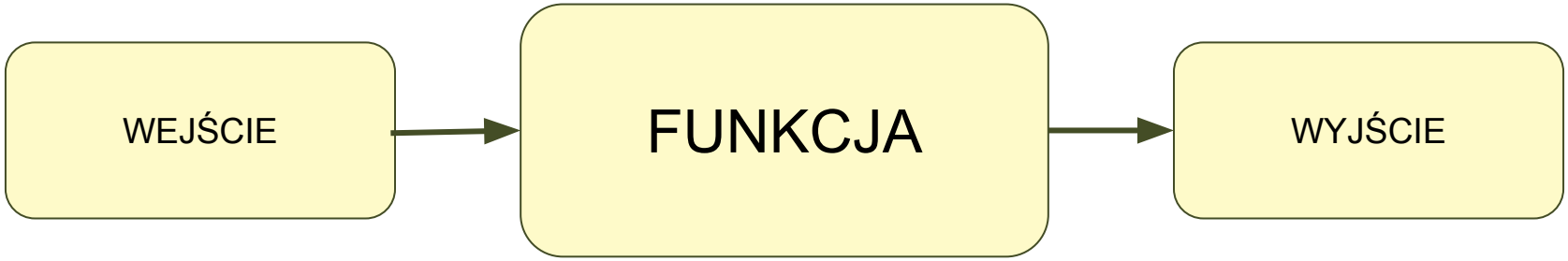
Funkcje i przepływ sterowania

Bartłomiej Tartanus

Funkcje

Funkcje

Funkcja ma wykonywać obliczenia i zwracać jakąś wartość, nie powinna natomiast mieć żadnego innego wpływu na działanie programu.



Definiowanie funkcji

```
function(list_parametrów) ciało_funkcji
```

Przykład funkcji liczącej kwadrat:

```
> (function(x) x^2)(1:5)
[1]  1  4  9 16 25

> f <- function(x,y) x^y; f(1:5, 2)
[1]  1  4  9 16 25
```

Zwracanie wartości

```
domyslnie <- function(x){
   2 * x
   x + 5
}
> y <- domyslnie(1:3)
> y
[1] 6 7 8
```

Zwracanie wartości

```
przezReturn <- function(x){
   return(2 * x)
   x + 5
}
> przezReturn(1:3)
[1] 2 4 6
```

Funkcje z zewnątrz

```
install.packages("stringi")
require("stringi")

stri_paste("A",1:5)

source("/ścieżka/do/pliku.R")
```

Zasięg nazw w funkcjach

Zmienne lokalne

```
f1 <- function(x){
   y <- x + 1
   cat("x=",x," y=",y)
}
> f1(x = 3)
x= 3  y= 4
```

```
> x
BŁĄD: nie znaleziono
 obiektu 'x'
> y
BŁĄD: nie znaleziono
 obiektu 'y'
```

Zasięg nazw w funkcjach

Zmienne spoza funkcji

```
f2 <- function(){
  cat("x=",x,"\n")
  x <- 5
  cat("x=",x,"\n")
}
```

```
> x <- 2
> f2()
x= 2
x= 5
> x
[1] 2
```

Zasięg nazw w funkcjach

Nie zalecamy odwoływać się do zmiennych spoza funkcji jeśli nie zostały one przekazane jako jeden z argumentów, gdyż może to powodować błędy trudne do wyłapania.

Przekazywanie argumentów przez wartość

```
zwieksz <- function(x){
   x <- 2*x
   x
}
> y <- 1:5
> zwieksz(y)
[1]  2  4  6  8 10
> y
[1] 1 2 3 4 5
```

Parametry z argumentami domyślnymi

```
domyslne <- function(x=1, y=3){
  cat("x=",x,"y=",y)
}
```

```
> domyslne(4,4)
x= 4 y= 4
> domyslne(x=5)
x= 5 y= 3
```

```
> domyslne(y=5)
x= 1 y= 5
> domyslne()
x= 1 y= 3
```

Parametr specjalny ...

Pozwala przekazać dowolną liczbę argumentów:

```
sprawdz <- function(...){
  list(...)
}
> str(sprawdz(1:5, "abc", TRUE))
List of 3
 $ : int [1:5] 1 2 3 4 5
 $ : chr "abc"
 $ : logi TRUE
```

Parametr specjalny ...

Pozwala przekazać argumenty dalej:

```
zaokraglijSume <- function(ileCyfr, ...){
  round(sum(...), digits=ileCyfr)
}
> zaokraglijSume(0, 1.3:5, 1.123:5)
[1] 22
> zaokraglijSume(2, 1.3:5, 1.123:5)
[1] 21.69
```

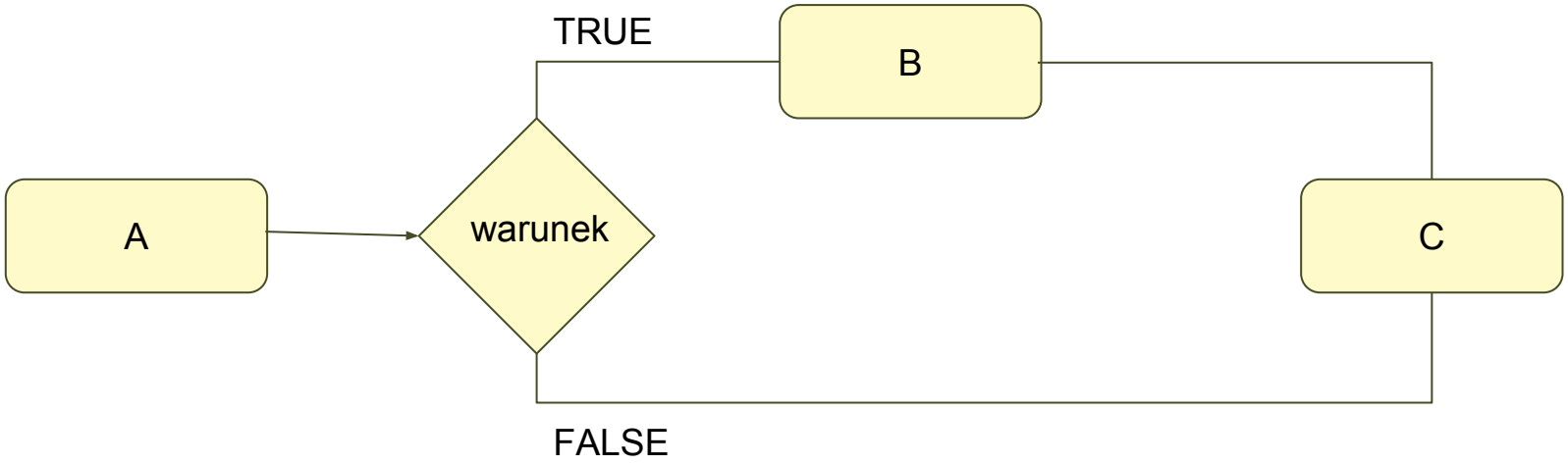
Leniwa ewaluacja

```
licz <- function(){
   cat("No to liczę")
}
wywolaj <- function(x){
   cat("1"); cat(x); cat("2"); cat(x); cat("3");
}
> wywolaj(licz())
1No to liczę23
```

Instrukcje warunkowe i pętle

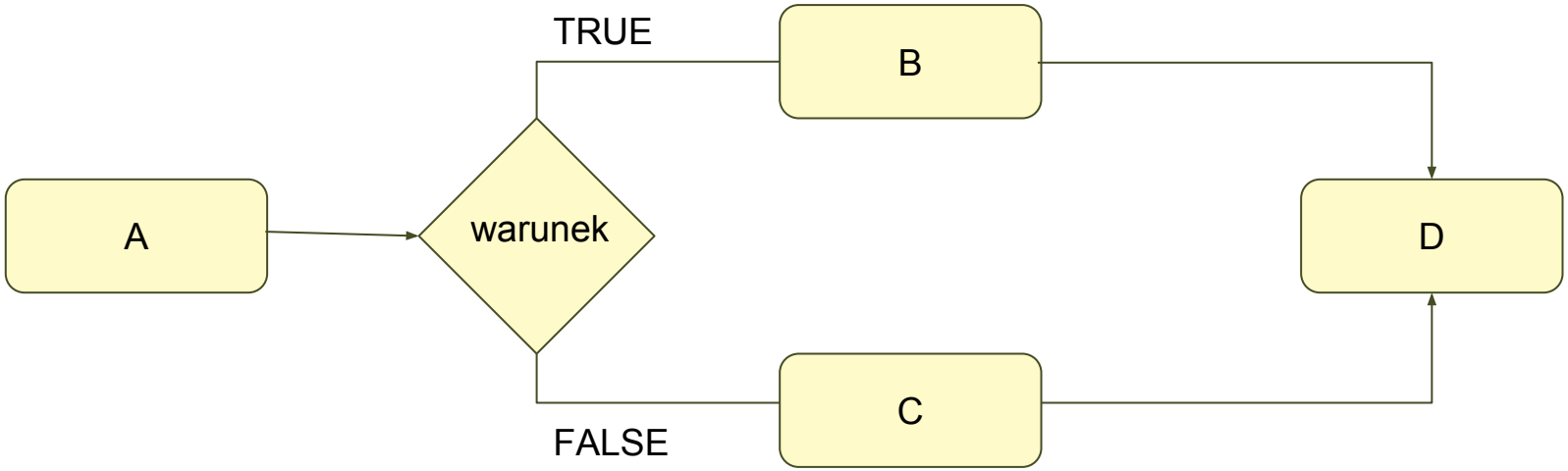
Instrukcja warunkowa `if`

```
A
if(warunek) wyrażenie_B
C
```



Instrukcja warunkowa `if else`

```
A
if(warunek) wyrażenie_B else wyrażenie_C
D
```



Sprawdzanie warunków

Funkcja `stopifnot()` pozwala zatrzymać działanie programu jeśli przekazane do niej warunki nie będą mieć wartości `TRUE`

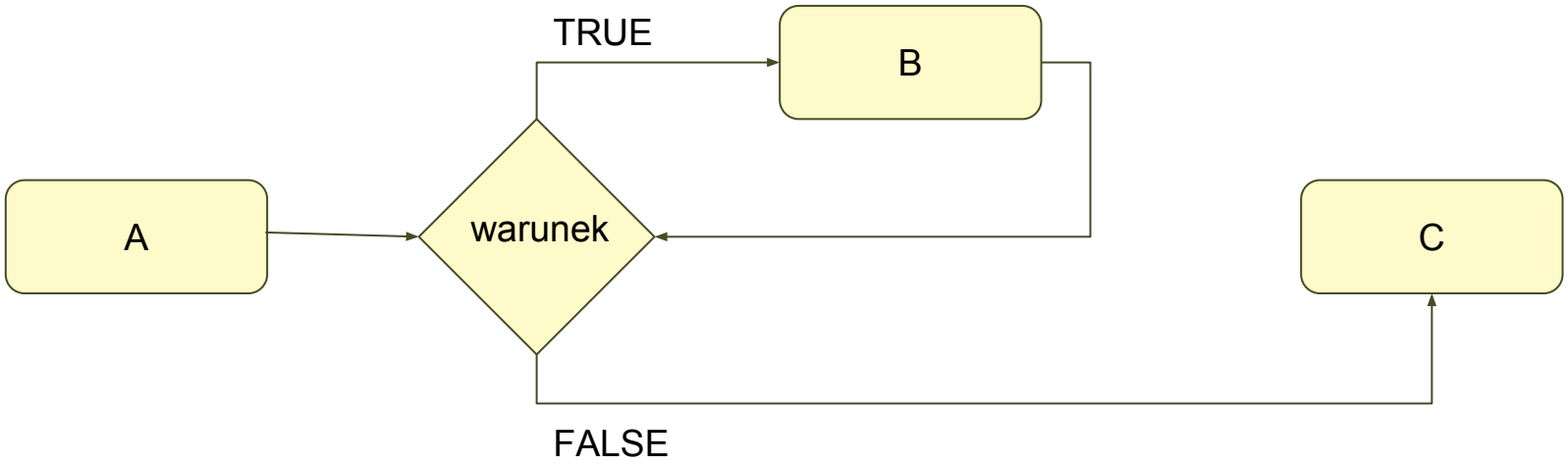
```
bezpiecznaSuma <- function(x){
   stopifnot(is.numeric(x))
   sum(x)
}
> bezpiecznaSuma("abc")
BŁĄD: zmienna is.numeric(x) nie ma wartości TRUE
```

Rekurencja

```
silnia <- function(n){
  if(n == 0 || n == 1){
    return(1)
  }else{
    return(n * silnia(n-1))
  }
}
> silnia(5)
[1] 120
```

Pętla `while`

```
A
while(warunek) wyrażenie_B
C
```



Pętla `for`

```
for(zmienna in wektor) wyrażenie
```

```
> for(i in 1:3) cat(i, " obrót\n")
1  obrót
2  obrót
3  obrót
```

Pętla `for` - pułapka

```
> wyrazy <- c("Ala ", "ma ", "kota ")
> for(i in 1:length(wyrazy)) cat(wyrazy[i])
Ala ma kota

> wyrazy <- c()
> for(i in 1:length(wyrazy)) cat(wyrazy[i])
## Co wtedy?
```

Pętla `for` - rozwiązanie

Używać `seq_along` lub `seq_len`

```
> seq_along(wyrazy)
[1] 1 2 3
> seq_along(c())
integer(0)
> seq_len(5)
[1] 1 2 3 4 5
> seq_len(length(wyrazy))
[1] 1 2 3
```

Wydajność pętli

Pętle w R są **WOLNE** i należy ich unikać

(oczywiście jeśli to możliwe)

```
suma <- 0
for(i in seq_along(x))
  suma <- suma + x[i]

sum(x)
```