

# base R, pt.2

# Вектори TL;DR

Вектор є базовим типом структури у R. Скаляри насправді є вектором з довжиною один.

Вектори бувають:

- атомарні, що містять гомогенні дані одного типу
  - **character** — текстові значення, мають бути заключені у " " або ' '
  - **logical** — логічні значення
  - **integer** — цілочислові значення
  - **double** — число з плаваючою комою (у панелі середовища відображено як **numeric**)
  - **complex** — комплексне число типу  $n + i$
  - **raw** — для представлення "сирої" послідовності байтів
- листи, що можуть містити гетерогенні дані різних типів та інші листи

Створити пустий вектор можливо командою **vector()** або командою, що відповідає одному з типів описаних вище, визначити тип командою **typeof()**, визначити довжину командою **length()**

```
1 x <- vector(mode = "logical", length = 0L) # теж саме що x <- logical(length = 0L)
2 typeof(x)
3 #> [1] "logical"
4 length(x)
5 #> [1] 0
```

Чотири основні типи векторів, **complex** та **raw** зустрічаються відносно рідко і не показані тут

```
1 char <- c("This is a String", "This is a second String")
2 print(char)
3 #> [1] "This is a String"          "This is a second String"
4 typeof(char)
5 #> [1] "character"
6 length(char) # зверніть увагу на довжину, вектор містить два елементи
7 #> [1] 2
8
9 int <- 1:3
10 typeof(int)
11 #> [1] "integer"
12 also_int <- c(1L, 2L, 3L) # суфікс L експліцитно означає цілочислові значення
13 typeof(also_int)
14 #> [1] "integer"
15
16 dbl <- c(1.00, 2.5, 0.0002)
17 typeof(dbl)
18 #> [1] "double"
19 also_dbl <- c(1, 2, 3)
20 typeof(also_dbl)
21 #> [1] "double"
22
23 logic <- c(TRUE, FALSE) # R також допускає скорочення T та F
24 typeof(logic)
25 #> [1] "logical"
```

Оскільки атомарні вектори можуть містити лише об'єкти одного типу, при конкатенації векторів різних типів буде виконана спроба конверсії одного типу у інший

Об'єднання будь-якого типу з типом **character** конвертує значення у текстові:

```
1 new_vector <- c(char, logic, dbl)
2 print(new_vector)
3 #> [1] "This is a String"      "This is a second String"
4 #> [3] "TRUE"                      "FALSE"
5 #> [5] "1"                        "2.5"
6 #> [7] "2e-04"
7 typeof(new_vector)
8 #> [1] "character"
```

Об'єднання типу **logical** з чисельними значеннями конвертує логічні значення у чисельні:

```
1 c(logic, int)
2 #> [1] 1 0 1 2 3
3 c(TRUE, TRUE, 5, FALSE, FALSE)
4 #> [1] 1 1 5 0 0
```

Об'єднання типу **integer** з **double** дає **double**. Як факт, виконання будь-яких математичних операцій між **integer** та **double** поверне значення у форматі **double**:

```
1 typeof(1L + 2)
2 #> [1] "double"
```

Окрім того, R здатен зберігати у форматі **integer** лише значення від -214 748 3647 і до 214 748 3647. **x <- 2147483648L** буде збережено як **double** з відповідним попередженням

Спробувати “наси́льно” перевести вектор з одного типу до іншого, або перевірити чи є вектор вектором конкретного типу можливо командою `as.*(x, ...)` та `is.*(x)` де `*` замінено на відповідний тип

```
1 x <- c(TRUE, FALSE)
2 is.double(x)
3 #> [1] FALSE
4 as.double(x)
5 #> [1] 1 0
6
7 as.logical(c("true", "F", "bbbb")) # у випадку неможливості конверсії дає NA
8 #> [1] TRUE FALSE NA
```

Функції `numeric()` та `as.numeric()` дають такий же результат як `double()` та `as.double()`, проте `is.numeric()` загалом перевіряє чи може об’єкт бути інтерпретований як чисельний

```
1 is.numeric(1)
2 #> [1] TRUE
3 is.numeric(1L)
4 #> [1] TRUE
5 is.double(1L)
6 #> [1] FALSE
```

# Перетворення логічних векторів

Так як логічні значення **TRUE** та **FALSE** мають загальноприйняті відповідні їм чисельні значення **1** та **0**, значна кількість функцій, що виконують дії над чисельними векторами здатні також приймати вектори логічні, автоматично перетворивши їх

Це дозволяє робити так:

```
1 set.seed(5839)
2 logic <- sample(c(TRUE, FALSE), 623, replace = TRUE)
3
4 mean(logic)
5 #> [1] 0.4847512
6
7 sd(logic)
8 #> [1] 0.500169
9
10 sum(logic)
11 #> [1] 302
```

# Іменовані вектори

Елементи у векторах можуть мати імена, які можуть бути присвоєні безпосередньо при створенні

```
1 named_vec <- c(a = 1, b = 2, c = 3)
2 named_vec
3 #> a b c
4 #> 1 2 3
```

Або для вже існуючого вектору, командою `names(x) <-` або `setNames()`

```
1 names(named_vec) <- c("a", "b", "c")
2 named_vec
3 #> a b c
4 #> 1 2 3
5 named_vec <- setNames(named_vec, c("x", "y", "z"))
6 named_vec
7 #> x y z
8 #> 1 2 3
```

Переглянути імена елементів об'єкту, його інші атрибути та структуру можливо наступними командами:

```
1 str(named_vec) # структура об'єкту
2 #> Named num [1:3] 1 2 3
3 #> - attr(*, "names")= chr [1:3] "x" "y" "z"
4 attributes(named_vec) # атрибути об'єкту
5 #> $names
6 #> [1] "x" "y" "z"
7 attr(named_vec, "names") # тільки імена
8 #> [1] "x" "y" "z"
```

# Листи

Пустий лист бажаної довжини також може бути створений командою тією ж командою `vector()`

```
1 empty_ls <- vector(mode = "list")
2 empty_ls # пустий лист довжиною 0
3 #> list()
```

Сконструювати лист можливо командою `list(...)`

```
1 some_ls <- list(c(1, 2, 3), 5.0, "a")
2 some_ls
3 #> [[1]]
4 #> [1] 1 2 3
5 #>
6 #> [[2]]
7 #> [1] 5
8 #>
9 #> [[3]]
10 #> [1] "a"
11
12 other_ls <- list(nums = 1:3, lettrs = c("a", "a", "b"), logic = T)
13 other_ls
14 #> $nums
15 #> [1] 1 2 3
16 #>
17 #> $lettrs
18 #> [1] "a" "a" "b"
19 #>
20 #> $logic
21 #> [1] TRUE
```



## Лист може мати лист у середні себе

```
1 bottomless_pit <- list(  
2   another_list = list(  
3     and_other_list = list(  
4       and_other_other_list = list(  
5         and_other_other_other_list = list(  
6           look_inside = "not bottomless",  
7           pit = "just regular pit now"  
8         ))))  
9  
10 bottomless_pit  
11 #> $another_list  
12 #> $another_list$and_other_list  
13 #> $another_list$and_other_list$and_other_other_list  
14 #> $another_list$and_other_list$and_other_other_list$and_other_other_other_list  
15 #> $another_list$and_other_list$and_other_other_list$and_other_other_other_list$look_inside  
16 #> [1] "not bottomless"  
17 #>  
18 #> $another_list$and_other_list$and_other_other_list$and_other_other_other_list$pit  
19 #> [1] "just regular pit now"
```

```
1 length(bottomless_pit)  
2 #> [1] 1  
3 str(bottomless_pit)  
4 #> List of 1  
5 #> $ another_list:List of 1  
6 #> ..$ and_other_list:List of 1  
7 #> .. ..$ and_other_other_list:List of 1  
8 #> .. .. ..$ and_other_other_other_list:List of 2  
9 #> .. .. .. ..$ look_inside: chr "not bottomless"  
10 #> .. .. .. ..$ pit : chr "just regular pit now"
```

## Сабсетинг векторів

**Сабсетинг** — “витягування” певного елементу структури, може бути виконане у шість різних способів із застосуванням трьох різних операторів, `[`, `[[` та `$` та може бути комбіноване з `<-`

Оператори `[[` та `$` повертають єдиний елемент. Оператор `$` не допускається до використання з атомарними векторами, але для інших структур може використовуватися для доступу до елементу по імені

Найпростіший варіант сабсетингу вектору — по індексу (позиції) елементу, позитивним цілим числом:

```
1 vec <- c(2.1, 5.2, 2.3, 1.4)
2 vec[[3]] # повернути елемент у позиції 3
3 #> [1] 2.3
4 vec[3]   # для атомарних векторів [i] та [[i]] фактично дають однаковий результат
5 #> [1] 2.3
6
7 vec[1:3] # елементи з позиції 1 по 3
8 #> [1] 2.1 5.2 2.3
9 vec[c(2, 4)] # елемент у позиції 2 та 4
10 #> [1] 5.2 1.4
11
12 # менш інтуїтивні приклади
13 vec[c(4, 2, 1)]
14 #> [1] 1.4 5.2 2.1
15 vec[c(2, 2, 2, 2, 2)]
16 #> [1] 5.2 5.2 5.2 5.2 5.2
```

Сабсетинг негативним цілим числом повертає усі елементи крім вказаного:

```
1 vec[-3]
2 #> [1] 2.1 5.2 1.4
3 vec[-c(1, 3)] # теж саме що vec[c(-1, -3)]
4 #> [1] 5.2 1.4
5 vec[-c(1:3)]
6 #> [1] 1.4
```

Сабсетинг текстом дозволяє витягнути елемент по його імені:

```
1 names(vec) <- c("one", "two", "three", "four")
2
3 vec[["two"]]
4 #> [1] 5.2
5 vec[c("one", "one")]
6 #> one one
7 #> 2.1 2.1
```

Сабсетинг логічними значеннями повертає елементи позиція яких відповідає значенню TRUE:

```
1 vec[c(TRUE, FALSE, TRUE, FALSE)]
2 #> [1] 2.1 2.3
3 vec[c(FALSE, TRUE)] # ресайклінг до FALSE TRUE FALSE TRUE
4 #> [1] 5.2 1.4
5
6 vec[TRUE]
7 #> [1] 2.1 5.2 2.3 1.4
8 vec[[TRUE]]
9 #> [1] 2.1
```

Ніщо повертає оригінальний вектор:

```
1 vec[]  
2 #> [1] 2.1 5.2 2.3 1.4
```

Нуль повертає вектор довжиною нуль:

```
1 vec[0]  
2 #> numeric(0)
```

Об'єкт, що подається до `[]` є звичайним вектором, що дозволяє використовувати для сабсетингу значення, що зберігаються у інших векторах, подавати до `[]` функції, якщо дані функції повертають вектор логічного, чисельного чи текстового типу та виконувати у `[]` логічні та арифметичні операції:

```
1 y <- c(1, 2, 2, 3, 4)  
2 vec[y]  
3 #> [1] 2.1 5.2 5.2 2.3 1.4  
4  
5 vec[2+2]  
6 #> [1] 1.4
```

Як вже було зазначено, можливо використовувати сабсетинг разом із `<-`

```
1 vec[[4]] <- 10  
2 vec[[5]] <- 5.5  
3 vec  
4 #> [1] 2.1 5.2 2.3 10.0 5.5  
5 vec[[1]] <- "a"  
6 vec  
7 #> [1] "a" "5.2" "2.3" "10" "5.5"
```

## which() та деякі інші команди

Функція `which(x)` повертає індекс елементів вектору, що задовольняють вказану логічну умову:

```
1 set.seed(2312)
2 x <- sample(100, 10)
3 x
4 #> [1] 71 44 95 13 74 41 17 85 10 35
5
6 which(x > 10 & x < 35)
7 #> [1] 4 7
8 which.max(x)
9 #> [1] 3
10 which.min(x)
11 #> [1] 9
```

Низка інших функцій для роботи з векторами

```
1 order(x) # повертає індекси сортованого вектору
2 #> [1] 9 4 7 10 6 2 1 5 8 3
3 sort(x) # повертає сортований вектор
4 #> [1] 10 13 17 35 41 44 71 74 85 95
5 rev(c(3, 10, 1)) # перегортає вектор
6 #> [1] 1 10 3
7 unique(rep(letters[5:9], 10)) # повертає унікальні значення
8 #> [1] "e" "f" "g" "h" "i"
9 table(rep(letters[5:9], 10)) # крос-табуляція, докладніше пізніше
10 #>
11 #> e f g h i
12 #> 10 10 10 10 10
```

# Сабсетинг листів

Майже як вектори, але `[` завжди повертає структуру типу лист

```
1 example_ls <- list(a = "a", b = 1:3, c = list("One", 2))
2 example_ls[1]
3 #> $a
4 #> [1] "a"
5 example_ls[3]
6 #> $c
7 #> $c[[1]]
8 #> [1] "One"
9 #>
10 #> $c[[2]]
11 #> [1] 2
12 typeof(example_ls[3])
13 #> [1] "list"
14 typeof(example_ls[1])
15 #> [1] "list"
```

Оператор `[[` повертає єдиний елемент з відповідним типом

```
1 example_ls[[1]] # референсинг по індексу
2 #> [1] "a"
3 example_ls[["b"]] # референсинг по імені
4 #> [1] 1 2 3
5 typeof(example_ls[[1]])
6 #> [1] "character"
7 typeof(example_ls[["b"]])
8 #> [1] "integer"
```

```
1 example_ls[["c"]]
2 #> [[1]]
3 #> [1] "One"
4 #>
5 #> [[2]]
6 #> [1] 2
7 typeof(example_ls[["c"]])
8 #> [1] "list"
9 example_ls[["c"]][[1]]
10 #> [1] "One"
11 typeof(example_ls[["c"]][[1]])
12 #> [1] "character"
```

Оператор **\$** фактично є скороченням оператора **[[** для доступу по імені

```
1 example_ls$a
2 #> [1] "a"
3 example_ls$c[[2]]
4 #> [1] 2
```

Як і у випадку з атомарним вектором, **<-** може бути використано для заміни або додавання елементів, окрім того **<-** може бути комбіноване з **NULL** для видалення елементу листа

```
1 example_ls$c <- NULL # це видалить елемент
2 example_ls$w <- list(NULL) # а це буквально додасть елемент зі значенням NULL
```

# Матриці

Прямокутні структури, що мають колонки та рядки. Фактично є векторами, що мають атрибут розмірності `dim`. Як і атомарні вектори, матриці можуть містити дані лише одного типу.

Створення матриці командою `matrix()`, дефолтно матриця будується по колонкам

```
1 m <- matrix(c(1, 2, 3,
2               1, 2, 3, # вказати кількість колонок або рядків, базове значення 1
3               1, 2, 3), nrow = 3)
4 m
5 #>      [,1] [,2] [,3]
6 #> [1,]    1    1    1
7 #> [2,]    2    2    2
8 #> [3,]    3    3    3
```

Можливо вказати `byrow=TRUE`, щоб матриця була побудована по рядках

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



# Матриці

```
1 attributes(m) # матриця 3x3
2 #> $dim
3 #> [1] 3 3
4 dim(m)
5 #> [1] 3 3
6 nrow(m) # кількість рядків
7 #> [1] 3
8 ncol(m) # кількість колонок
9 #> [1] 3
```

Звичайний атомарний вектор не має просторової розмірності взагалі

```
1 n <- 1:9
2 dim(n)
3 #> NULL
4
5 dim(n) <- c(1, 9) # тепер має
6 n
7 #>      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
8 #> [1,]    1    2    3    4    5    6    7    8    9
9
10 dim(n) <- c(3, 3)
11 n
12 #>      [,1] [,2] [,3]
13 #> [1,]    1    4    7
14 #> [2,]    2    5    8
15 #> [3,]    3    6    9
```

# Матриці

Так як матриці є векторами у розумінні R звичайні арифметичні та логічні операції дають такий самий результат як з векторами

```
1 m * n # елемент n[i, j] буде помножено на елемент m[i, j]
2 #>      [,1] [,2] [,3]
3 #> [1,]     1     8    21
4 #> [2,]     2    10    24
5 #> [3,]     3    12    27
```

Для виконання дій лінійної алгебри є низка своїх операторів, зокрема `%*%` для множення матриці

```
1 m %*% n
2 #>      [,1] [,2] [,3]
3 #> [1,]    14    32    50
4 #> [2,]    14    32    50
5 #> [3,]    14    32    50
```

```
1 crossprod(X, Y = NULL) # крос-продукт
2 tcrossprod(X, Y = NULL)
3 outer(X, Y) # зовнішній продукт
4 %o%
```

# Матриці

Матриця може бути транспонована командою `t(x)`

```
1 t(n)
2 #>      [,1] [,2] [,3]
3 #> [1,]    1    2    3
4 #> [2,]    4    5    6
5 #> [3,]    7    8    9
```

Колонки та рядки матриці можуть мати імена, які можуть бути встановлені відповідно командами `colnames()` та `rownames()`, аналогічно `names()` для векторів. Додати нову колонку або рядок до матриці можливо застосуванням `cbind()` та `rbind()`

```
1 n <- cbind(n, c(0, 0, 0))
2 rownames(n) <- LETTERS[1:3]
3 colnames(n) <- letters[1:4]
4 n
5 #>   a b c d
6 #> A 1 4 7 0
7 #> B 2 5 8 0
8 #> C 3 6 9 0
```

# Сабсетинг матриць

Як вектори, але мають додатковий вимір, тому у `[]` ідуть два значення, перше вказує на рядок, друге на колонку

```
1 n[[2, 3]] # другий елемент третьої колонки
2 #> [1] 8
3 n[1:2, 1:2]
4 #>      a b
5 #> A 1 4
6 #> B 2 5
7 n[2, ] # другий рядок повністю
8 #> a b c d
9 #> 2 5 8 0
10 n[, 3] # третя колонка повністю
11 #> A B C
12 #> 7 8 9
13 n[, 3, drop = FALSE] # збереження розмірності
14 #>      c
15 #> A 7
16 #> B 8
17 #> C 9
18
19 typeof(n) # матриця не є типом, як не дивно
20 #> [1] "double"
21 class(n)
22 #> [1] "matrix" "array"
23 class(n[, 3])
24 #> [1] "numeric"
25 class(n[, 3, drop = FALSE])
26 #> [1] "matrix" "array"
```

*P.s* Ви могли помітити, що об'єкт окрім класу `matrix` також мав клас `array` — масив. Окрім двовірних матриць можливо створювати структури вищої розмірності. Сабсетинг такий самий, просто з оглядом на додаткові виміри

```
1 arr <- array(1:18, c(3, 3, 2)) # простий приклад, може бути гірше
2 arr
3 #> , , 1
4 #>
5 #>      [,1] [,2] [,3]
6 #> [1,]    1    4    7
7 #> [2,]    2    5    8
8 #> [3,]    3    6    9
9 #>
10 #> , , 2
11 #>
12 #>      [,1] [,2] [,3]
13 #> [1,]   10   13   16
14 #> [2,]   11   14   17
15 #> [3,]   12   15   18
16
17 arr[1, 3, 2]
18 #> [1] 16
```

# Кадри даних (Data Frames)

Прямокутні структури, що мають колонки та рядки. Подібні за властивостями до листів, фактично кажучи колонки кадрів даних і є векторами листів — таким чином *колонки* мають включати у себе об'єкти одного типу, у той час як *рядки* можуть містити об'єкти різних типів.

Створити кадр даних можливо командою `data.frame()`, вказавши імена та значення для колонок:

```
1 df <- data.frame(  
2   char = c(letters[1:4]),  
3   int = 1:4,  
4   dbl = runif(4))  
5 df  
6 #>   char int      dbl  
7 #> 1    a   1 0.2103072  
8 #> 2    b   2 0.2104003  
9 #> 3    c   3 0.4291486  
10 #> 4    d   4 0.1698004
```

Краще зрозуміти як можуть виглядати кадри даних можливо через ознайомлення з класичними датасетами представленими у R, наприклад `mtcars` або `iris`. Або взяти щось менш популярне.

```
1 data(crabs, package = "MASS") # це експліцитно завантажить ці данні у Ваш робочій простір  
2 #?MASS::crabs щоб дізнатися більше про набір даних
```

Щоб інтерактивно переглянути кадр даних у більш звичному стилі електронної таблиці викличте `View(x)`

# Кадри даних

Швидке знайомство з кадром даних можливо провести переглянувши декілька перших рядків у консолі

```
1 head(crabs, n = 5)
2 #>   sp sex index  FL  RW  CL  CW  BD
3 #> 1  B  M     1 8.1 6.7 16.1 19.0 7.0
4 #> 2  B  M     2 8.8 7.7 18.1 20.8 7.4
5 #> 3  B  M     3 9.2 7.8 19.0 22.4 7.7
6 #> 4  B  M     4 9.6 7.9 20.1 23.1 8.2
7 #> 5  B  M     5 9.8 8.0 20.3 23.0 8.2
```

Або краще, переглянувши його структуру

```
1 str(crabs)
2 #> 'data.frame':    200 obs. of  8 variables:
3 #> $ sp      : Factor w/ 2 levels "B","O": 1 1 1 1 1 1 1 1 1 1 ...
4 #> $ sex     : Factor w/ 2 levels "F","M": 2 2 2 2 2 2 2 2 2 2 ...
5 #> $ index: int  1 2 3 4 5 6 7 8 9 10 ...
6 #> $ FL      : num  8.1 8.8 9.2 9.6 9.8 10.8 11.1 11.6 11.8 11.8 ...
7 #> $ RW      : num  6.7 7.7 7.8 7.9 8 9 9.9 9.1 9.6 10.5 ...
8 #> $ CL      : num  16.1 18.1 19 20.1 20.3 23 23.8 24.5 24.2 25.2 ...
9 #> $ CW      : num  19 20.8 22.4 23.1 23 26.5 27.1 28.4 27.8 29.3 ...
10 #> $ BD      : num  7 7.4 7.7 8.2 8.2 9.8 9.8 10.4 9.7 10.3 ...
```

# Кадри даних

Для приєднання нових колонок / рядків використовуються тіж команди, що і у випадку матриці

```
1 colnames(crabs) <- c(colnames(crabs)[1:3], "front_lobe", "rear_wd",  
2                       "carapace_l", "carapace_wd", "body_depth")  
3 crabs <- cbind(crabs, species = rep("crab", nrow(crabs)))  
4  
5 head(crabs, 5)  
6 #>   sp sex index front_lobe rear_wd carapace_l carapace_wd body_depth species  
7 #> 1  B  M     1      8.1      6.7      16.1      19.0      7.0      crab  
8 #> 2  B  M     2      8.8      7.7      18.1      20.8      7.4      crab  
9 #> 3  B  M     3      9.2      7.8      19.0      22.4      7.7      crab  
10 #> 4  B  M     4      9.6      7.9      20.1      23.1      8.2      crab  
11 #> 5  B  M     5      9.8      8.0      20.3      23.0      8.2      crab
```

Сабсетинг є дещо міксом між матрицями та листами, але зазвичай це не стає проблемою

```
1 small_crabs <- head(crabs, n = 3) # маленький шматок для демонстрації
```



## Сабсетинг кадрів даних

```
1 small_crabs[4:9]      # колонки з 4 по 9, сабсетинг як лист
2 #>   front_lobe rear_wd carapace_l carapace_wd body_depth species
3 #> 1         8.1      6.7      16.1      19.0      7.0      crab
4 #> 2         8.8      7.7      18.1      20.8      7.4      crab
5 #> 3         9.2      7.8      19.0      22.4      7.7      crab
6 small_crabs[, 4:9] # теж саме, сабсетинг як матриця
7 #>   front_lobe rear_wd carapace_l carapace_wd body_depth species
8 #> 1         8.1      6.7      16.1      19.0      7.0      crab
9 #> 2         8.8      7.7      18.1      20.8      7.4      crab
10 #> 3         9.2      7.8      19.0      22.4      7.7      crab
11 small_crabs[2, 1:4]
12 #>   sp sex index front_lobe
13 #> 2  B  M     2         8.8
14
15 str(small_crabs["index"]) # зберігає структуру
16 #> 'data.frame':      3 obs. of  1 variable:
17 #>  $ index: int   1 2 3
18 str(small_crabs[, "index"]) # симпліфікація до атомарного вектору, як і у випадку матриці
19 #>  int [1:3] 1 2 3
20 str(small_crabs[, "index", drop = FALSE])
21 #> 'data.frame':      3 obs. of  1 variable:
22 #>  $ index: int   1 2 3
23 str(small_crabs$index) # також дроп структури
24 #>  int [1:3] 1 2 3
```

## Сабсетинг кадрів даних

```
1  typeof(small_crabs["index"]) # можна впевнитися, що колонки кадрів дійсно є списками
2  #> [1] "list"
3  # як і з списками, NULL можна використати, щоб видалити колонку
4  small_crabs$species <- NULL
5  small_crabs
6  #>   sp sex index front_lobe rear_wd carapace_l carapace_wd body_depth
7  #> 1  B  M     1         8.1      6.7        16.1         19.0         7.0
8  #> 2  B  M     2         8.8      7.7        18.1         20.8         7.4
9  #> 3  B  M     3         9.2      7.8        19.0         22.4         7.7
10
11 # так само додати нову
12 small_crabs$new_val <- NA # NA для прикладу, сюди можна присвоїти значення одразу
13 small_crabs
14 #>   sp sex index front_lobe rear_wd carapace_l carapace_wd body_depth new_val
15 #> 1  B  M     1         8.1      6.7        16.1         19.0         7.0      NA
16 #> 2  B  M     2         8.8      7.7        18.1         20.8         7.4      NA
17 #> 3  B  M     3         9.2      7.8        19.0         22.4         7.7      NA
18
19 # базове практичне застосування з іншими функціями
20 mean(small_crabs$rear_wd)
21 #> [1] 7.4
22 max(small_crabs$front_lobe)
23 #> [1] 9.2
```