Note 3. Vector

Introduction to Statistical Programming

Chanmin Kim

Department of Statistics
Sungkyunkwan University

2022 Spring

Scalar & Vector

- Most languages (e.g., C, Python, etc.): Scalar $\neq$ Vector.
- R: Scalar $=$ Vector.
- Creating vectors:
  - `c()` function: Directly input vector elements.
  - `vector('format',length)` function: Initialized vectors.
- Length of vector: `length(vector)`.
  - `NULL` $\Rightarrow$ zero length.

Example: Creating Vectors & Length of Vectors

```
> x <- 7
> x
[1] 7
> is.vector(x)
[1] TRUE
> x <- c('a','b','c')
> x <- c(T,T,F,T)
> x <- vector('numeric',10)
> x
 [1] 0 0 0 0 0 0 0 0 0 0
> x <- vector('character',5)
> x
[1] "" "" "" "" ""
> length(x)
[1] 5
```

Vector Indexing

- Creating subvector by picking elements of the given vector:

```
> x <- c(2,-3,5,4,7,2,-1,2,9,0)
> x[c(2,4,6,8)]
[1] -3  4  2  2
> idx <- c(2,4,6,8)
> x[idx]
[1] -3  4  2  2
> 2:5
[1] 2 3 4 5
> a <- 2:5
> y <- x[2:5]; y
[1] -3  5  4  7
```

- Elimination of elements: Indexes with - sign.

```
> x[-(2:5)]
[1]  2  2 -1  2  9  0
> x[c(-2,-4)]
[1]  2  5  7  2 -1  2  9  0
```

Manipulation of Vectors

- Adding or deleting elements ⇒ Reassign the vector.

```
> x <- 1:5
> y <- c(x[1:3],3.5,x[4:5])
> y
[1] 1.0 2.0 3.0 3.5 4.0 5.0
>
> y <- y[-c(1,3,4)]
> y
[1] 2 4 5
```

- Matrices & arrays have the same properties of vectors ⇒ Most vector manipulations work for matrices & arrays.

```
> x <- matrix(2,2,2)
> length(x)
[1] 4
> x + 1:4
     [,1] [,2]
[1,]    3    5
[2,]    4    6
```

Manipulation of Vectors

- Replacement of vector elements:

```
> x <- 10:1
> x[5] <- 0
> x
 [1] 10  9  8  7  0  5  4  3  2  1
> x[5:10] <- 0
> x
 [1] 10  9  8  7  0  0  0  0  0  0
> x[5:10] <- c(2,3,4,5,6,7)
> x
 [1] 10  9  8  7  2  3  4  5  6  7
```

Vector Operations

- `+`, `-`, `*`, `/`, `%%`: Element-wise operations:

```
> x <- c(10,15,40); y <- c(5,2,3)
> x + y
[1] 15 17 43
> x - y
[1]  5 13 37
> x * y
[1]  50  30 120
> x / y
[1]  2.00000  7.50000 13.33333
> x %% y
[1] 0 1 1
> x + 2    # Recycling
[1] 12 17 42
> x - 2    # Recycling
[1]  8 13 38
> x * 2
[1] 20 30 80
> x / 2
[1]  5.0  7.5 20.0
```

Vector Operations

- Element-wise operators:

Operator	Description	Operator	Description
$\wedge$	Power	`exp()`	Exponential
`sqrt()`	Square root	`round()`	Round up
`log()`	Log with base e	`ceiling()`	Ceiling
`log10()`	Log with base 10	`floor()`	Floor

```
> x <- c(9, 2.2, 3.7)
> x^2
[1] 81.00  4.84 13.69
> sqrt(x)
[1] 3.000000 1.483240 1.923538
> round(x)
[1] 9 2 4
> ceiling(x)
[1] 9 3 4
> floor(x)
[1] 9 2 3
```

Useful Operators

- `seq(from, to, by)` or `seq(from, to, length)`: Generating a sequence in arithmetic progression.

```
> seq(1,10,by=2)
[1] 1 3 5 7 9
> seq(1,10,length=5)
[1]  1.00  3.25  5.50  7.75 10.00
> seq(10,1,by=-2)
[1] 10  8  6  4  2
```

- `seq(`*vector*`)` $\equiv$ `1:length(`*vector*`)`.
  - If the *vector* has zero length, then it returns `0`.
  - It is useful when the loop statement is used.

```
> x <- c(10,5,7); seq(x)
[1] 1 2 3
> x <- NULL; seq(x)
integer(0)
```

Useful Operators

- `rep(vector, times=b)`: Repeat the *vector* *b* times.

```
> rep(3,5)
[1] 3 3 3 3 3
> rep(c(2,4,6),3)
[1] 2 4 6 2 4 6 2 4 6
```

- `rep(vector, each=b)`: Repeat each element of the *vector* *b* times.

```
> rep(c(5,2,3),each=3)
[1] 5 5 5 2 2 2 3 3 3
```

NA

- NA: Not Available; Missing data.
  - NA's are counted as existing values
  - Many statistical R functions do NOT work for objects with NA elements.
  - Arguments in the functions to skip over NA elements.
  - NA can be in any mode of R objects.

```
> x = c(10,20,NA,30,40)
> length(x)
[1] 5
> mean(x)
[1] NA
> mean(x,na.rm=T)
[1] 25
> mode(x)
[1] "numeric"
> x = c(T,F,NA,T,F,F); mode(x)
[1] "logical"
> x = c('a',NA,'c'); mode(x)
[1] "character"
```

NULL

- NULL: Values do NOT exist.
  - NULL's are NOT counted.
  - Statistical functions work for objects with NULL's.
  - It can be used as an initialized object.

```
> x <- c(10,20,NULL,30,40)
> mean(x)
[1] 25
> length(x)
[1] 4
>
> x <- NULL
> for (i in 1:10) x <- c(x,i)
> x
 [1]  1  2  3  4  5  6  7  8  9 10
```

Logical Operators

- Logical operators:

Operator	Description
`x ==` *value*	Equal to *value*.
`x !=` *value*	Not equal to *value*.
`x <` *value*	Less than *value*.
`x >` *value*	Greater than *value*.
`x <=` *value*	Less than or equal to *value*.
`x >=` *value*	Greater than or equal to *value*.
`x == y`	Comparison for element-wise equality.
`x != y`	Comparison for element-wise not equality.
`x > (>=) y`	Comparison for element-wise inequality.
`x < (<=) y`	Comparison for element-wise inequality.

  - NOTE: `x & y` are vector objects.

Logical Operators

- E.g., logical operators:

```
> x <- 1:7; x == 4
[1] FALSE FALSE FALSE  TRUE FALSE FALSE FALSE
> x != 4
[1]  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE
> x > 4
[1] FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE
> x <- c('kim','lee','park','choi')
> x == 'park'
[1] FALSE FALSE  TRUE FALSE
> x != 'park'
[1]  TRUE  TRUE FALSE  TRUE
> x < 'kim'     # Comparison by UTF-8 code.
[1] FALSE FALSE FALSE  TRUE
> x <- c(1,5,10); y = c(3,5,7)
> x == y
[1] FALSE  TRUE FALSE
> x >= y
[1] FALSE  TRUE  TRUE
```

all() & any() Functions

- all():
  - At least one FALSE $\Rightarrow$ FALSE.
  - All TRUE $\Rightarrow$ TRUE.

```
> x <- 1:7
> all(x > 4)
[1] FALSE
> all(x > 0)
[1] TRUE
```

- any():
  - At least one TRUE $\Rightarrow$ TRUE.
  - All FALSE $\Rightarrow$ FALSE.

```
> x <- 1:7
> any(x > 4)
[1] TRUE
> any(x > 10)
[1] FALSE
```

identical() Function

- identical(*object*, *object*):
  - Comparison for equality of two R objects (vector, matrix, array, list, data frame, etc.).
  - If everything such as value, mode, type, etc. is the same, it returns TRUE.

```
> x <- 1:3
> y <- c(1,2,3)
> all(x == y)
[1] TRUE
> identical(x,y)
[1] FALSE
>
> typeof(x)
[1] "integer"
> typeof(y)
[1] "double"
```

Filtering

- Filtering: Extracting elements satisfying certain conditions.
- Filtering indexes: Extracting elements by indexes.

```
> x <- c(-2,7,5,0,-10)
> y <- x[x > 3]
> y
[1] 7 5
> x[c(F,T,T,F,F)]
[1] 7 5
```

- Assignment using filtering indexes: Replacing elements by filtering indexes.

```
> x[x > 3] <- 10
> x
[1]  -2  10  10   0 -10
```

Filtering

- `subset()`: This function works in the same manner as the filtering indexes except for handling `NA`.

```
> x <- c(-2,7,NA,5,0,NA,-10)
> x[x > 3]
[1]  7 NA  5 NA
> subset(x,x>3)
[1] 7 5
```

- `which()`: Indexes of elements satisfying certain conditions.

```
> x <- c(-2,7,5,0,-10)
> which(x > 3)
[1] 2 3
>
> x <- c('kim','lee','park','choi')
> which(x == 'park')
[1] 3
```

ifelse() Function

- ifelse(*condition, value for T, value for F*):
  - A vector version of if-then-else statement.
  - It returns a vector with the same length as the vector in *condition*.

```
> x <- 1:10
> y <- ifelse(x %% 3 == 0, 0, 1)
> y
 [1] 1 1 0 1 1 0 1 1 0 1
>
> x <- c('A','A','B','C','C','A')
> ifelse(x == 'A', 1, 0)
[1] 1 1 0 0 0 1
```

sort() & order() Functions

- sort(*vector*,decreasing=F):
  - Sort elements of a vector in either increasing or decreasing order.
  - It returns the result of the sorted vector.

- order(*vector*,decreasing = F):
  - It returns the original indexes of sorted elements.

```
> x <- c(3,1,7,5)
> sort(x)
[1] 1 3 5 7
> sort(x,decreasing=T)
[1] 7 5 3 1
> order(x)
[1] 2 1 4 3
> x <- c('kim','park','lee','choi')
> sort(x)
[1] "choi" "kim"  "lee"  "park"
> order(x)
[1] 4 1 3 2
```

Vector Element Names

- Name can be assigned to each element of vector using `names()` function.
- Indexing by name is possible.
- Name can be removed by assigning `NULL`.

```
> x <- 1:3
> names(x) <- c('a','b','c')
> x
a b c
1 2 3
> x[c('a','c')]
a c
1 3
> names(x) <- NULL
> x
[1] 1 2 3
```