Vectors in R

What's Vector?

Vector: a collection of same data type. It is called **Atomic structure**. Use `c()` to create a vector to combine multiple elements.

Examples

```
x <- 5 # a vector with length 1
x
```

```
## [1] 5
```

```
1 + 2 # addition of vectors with length 1
```

```
## [1] 3
```

```
c("a", "b", "c") # a vector of character
```

```
## [1] "a" "b" "c"
```

Most programming languages has a concept of scalar, but R doesn't.

Name Elements

```
x <- c("name" = "Vivi", "age" = 26, "sex" = "F")
x
```

```
##   name    age    sex
## "Vivi"   "26"    "F"
```

```
# obtain just names
names(x)
```

```
## [1] "name" "age"  "sex"
```

```
# change names
names(x) <- c("NAaaaaME", "AGE", "SEX")
x
```

```
## NAaaaaME      AGE      SEX
##   "Vivi"     "26"      "F"
```

```r
# name becomes NA if not defined
names(x) <- c("naaaaammmeeee")
x
```

```
## naaaaammmeeee          <NA>          <NA>
##        "Vivi"          "26"           "F"
```

Accessing an Element of Vector

Use brackets `[]` after the variable.

```r
x <- 1:10 # 1 2 3 4 5 6 7 8 9 10
x[1]
```

```
## [1] 1
```

```r
x[3]
```

```
## [1] 3
```

```r
x[1:2]
```

```
## [1] 1 2
```

```r
x[-2]
```

```
## [1]  1  3  4  5  6  7  8  9 10
```

```r
x[x %% 2 == 0] # elements whose remainder is 0 when divided by 2
```

```
## [1]  2  4  6  8 10
```

```r
names(x) <- c("a", "b", "c", "d", "e", "f", "g", "h", "i", "j")
x["c"] # access by name
```

```
## c
## 3
```

```r
x[c(1, 3, 8)] # obtain elements of index 1, 3, and 8
```

```
## a c h
## 1 3 8
```

Adding Elements

1. `name of vector[new index] <- element`

```r
x <- 1:3
x
```

```
## [1] 1 2 3
```

```r
x[4] <- 100
x
```

```
## [1]   1   2   3 100
```

```r
x[6] <- -100
x # skip undefined index and the undefined element becomes NA
```

```
## [1]    1    2    3  100   NA -100
```

2. use `c()`

```r
x
```

```
## [1]    1    2    3  100   NA -100
```

```r
x <- c(x, 22, Inf)
x
```

```
## [1]    1    2    3  100   NA -100   22  Inf
```

```r
# with name
y <- c("a" = 11, "b" = 45, "c" = -1)
y[4] <- 30
y # 30 has no name
```

```
##  a  b  c
## 11 45 -1 30
```

```r
y<- c(y, 100, 3000)
y # 30, 100, and 3000 have no name
```

```
##    a    b    c
##   11   45   -1   30  100 3000
```

Some Functions for Vector

- `length()`: returns a number of elements
- `sort()`: change the order of elements in increasing order. set `decreasing = TRUE` to order decreasingly
- `rev()`: reverse the order of elements
- `unique()`: returns only unique elements

```r
x <- c(1, 1, 2, 5, -9, 4, 2)
length(x)
```

```
## [1] 7
```

```r
sort(x)
```

```
## [1] -9  1  1  2  2  4  5
```

```r
sort(x, decreasing = TRUE)
```

```
## [1]  5  4  2  2  1  1 -9
```

```r
rev(x)
```

```
## [1]  2  4 -9  5  2  1  1
```

```r
unique(x)
```

```
## [1]  1  2  5 -9  4
```

```r
x # they don't mutate the original
```

```
## [1]  1  1  2  5 -9  4  2
```

Generate Sequences

1. :
2. seq()
3. rep()

```r
-2:5
```

```
## [1] -2 -1  0  1  2  3  4  5
```

```r
10:1
```

```
##  [1] 10  9  8  7  6  5  4  3  2  1
```

```r
-1:-5
```

```
## [1] -1 -2 -3 -4 -5
```

```r
-5:-1
```

```
## [1] -5 -4 -3 -2 -1
```

```r
seq(1, 10)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```r
seq(from = 1, to = 10, by = 2)
```

```
## [1] 1 3 5 7 9
```

```r
rep(1, times = 3)
```

```
## [1] 1 1 1
```

```r
rep(c(1, 2), times = 3)
```

```
## [1] 1 2 1 2 1 2
```

```r
rep(1:2, each = 3)
```

```
## [1] 1 1 1 2 2 2
```

```r
rep(1:2, length.out = 5)
```

```
## [1] 1 2 1 2 1
```

```r
rep(c(3, 2, 1), times = 3, each = 2)
```

```
##  [1] 3 3 2 2 1 1 3 3 2 2 1 1 3 3 2 2 1 1
```

```r
rep(seq(1, 5, 2), times = 2, each = 3)
```

```
##  [1] 1 1 1 3 3 3 5 5 5 1 1 1 3 3 3 5 5 5
```

Vectorization

Applying a function on all elements of vector.

```r
x <- 1:3
log_x <- log(x)
log_x
```

```
## [1] 0.0000000 0.6931472 1.0986123
```

```r
x / 2
```

```
## [1] 0.5 1.0 1.5
```

Recycling

Recycle the elements of vector of shorter length when operating something with vector of different length.

```
x <- 1:5
y <- 2:6 # same length as x
z <- 10 # vector of length 1

x + y # same length
```

```
## [1]  3  5  7  9 11
```

```
x + z # different length
```

```
## [1] 11 12 13 14 15
```

```
y - z # different length
```

```
## [1] -8 -7 -6 -5 -4
```

```
x * y # same length
```

```
## [1]  2  6 12 20 30
```

```
x * z # different length
```

```
## [1] 10 20 30 40 50
```

Data Types

There are 3 functions to check a data type: * `typeof()` returns a basic type of the variable * vector (integer, double, character, etc), list, function, ... * `mode()` returns a type of the element * numeric (integer, double), complex, logical, ... * `class()` returns a type of class of the object * matrix, array, factor, ... * if class is not defined, it returns a basic type or mode.

- logical: `TRUE`, `FALSE`, `T`, `F`, `NA`

```
typeof(TRUE)
```

```
## [1] "logical"
```

```
typeof(NA)
```

```
## [1] "logical"
```

- integer: to specify, add `L` next to a number; otherwise it is treated as "double" but it is rare to specify integer.

```r
typeof(1L)
```

```
## [1] "integer"
```

```r
typeof(1) # no "L" is treated as "double"
```

```
## [1] "double"
```

```r
mode(1L)
```

```
## [1] "numeric"
```

```r
class(1L)
```

```
## [1] "integer"
```

- double: a number with decimal or real number.

```r
typeof(5)
```

```
## [1] "double"
```

```r
typeof(-5)
```

```
## [1] "double"
```

```r
typeof(-0.3)
```

```
## [1] "double"
```

```r
typeof(1e-5) # e^-5
```

```
## [1] "double"
```

- complex: a complex number by adding "i".

```r
typeof(3 + 5i)
```

```
## [1] "complex"
```

```r
typeof(1i)
```

```
## [1] "complex"
```

```r
typeof(3 + 0i)
```

```
## [1] "complex"
```

```r
typeof(complex(re = 2, im = 3)) # 2 + 3i
```

```
## [1] "complex"
```

- character: string(s). it is always a character if it is surrounded by "".

```r
typeof("a")
```

```
## [1] "character"
```

```r
typeof("2020/09/10")
```

```
## [1] "character"
```

```r
typeof("")
```

```
## [1] "character"
```

```r
typeof(" ")
```

```
## [1] "character"
```

```r
typeof("NA") # missing value
```

```
## [1] "character"
```

Special Values

They usually remain the same after operation.

- NULL: a vector of no elements of length 0.

```r
typeof(NULL)
```

```
## [1] "NULL"
```

- NA: a missing value.

```r
typeof(NA)
```

```
## [1] "logical"
```

- Inf, -Inf: infinite number.

```r
typeof(Inf)
```

```
## [1] "double"
```

- `NaN`: not a number.

```r
typeof(NaN)
```

```
## [1] "double"
```

Example

```r
x <- c(1.0, NA, 3, 4.0)
x == NA # NA is applied to each element, and it returns just NA for each element
```

```
## [1] NA NA NA NA
```

To check special values, use the following functions:
- `is.null()` - `is.na()` - `is.finite()` - `is.infinite()` - `is.nan()`

Coercion

1. Explicit Coercion
   change the data type of a vector, if possible.

- `as.logical()`
- `as.integer()`
- `as.double()`
- `as.real()`
- `as.character()`

```r
x <- c("1", "2", "3.0")
typeof(x)
```

```
## [1] "character"
```

```r
as.integer(x) # no "L" but integer
```

```
## [1] 1 2 3
```

```r
as.double(x)
```

```
## [1] 1 2 3
```

```r
as.complex(x)
```

```
## [1] 1+0i 2+0i 3+0i
```

```r
x # they don't change the original vector
```

```
## [1] "1"   "2"   "3.0"
```

```r
# the following gives an warning and returns NA if the element is not convertible.
y <- c("1", "abc", 3, "def", NA, Inf, "7.0")
as.integer(y)
```

```
## Warning: NAs introduced by coercion
```

```
## Warning: NAs introduced by coercion to integer range
```

```
## [1]  1 NA  3 NA NA NA  7
```

```r
# this won't work because of "implicit coercion"
as.logical(x) # just returns NA for each element
```

```
## [1] NA NA NA
```

2. Implicit Coercion
   Hierarchy: **logical** $<$ **numeric (integer** $<$ **double)** $<$ **complex** $<$ **character**

```r
# different data types "double" and "complex" and "complex" dominates "double"
typeof(c(1, 1i))
```

```
## [1] "complex"
```

```r
z <- c(TRUE, FALSE, T, F, NA) # all elements are logical
z[2] <- -10
z # logical -> numeric
```

```
## [1]   1 -10   1   0  NA
```

```r
z[3] <- 0.5
z # integer -> double
```

```
## [1]   1.0 -10.0   0.5   0.0    NA
```

```r
z[4] <- 0.05
z # align decimal numbers
```

```
## [1]   1.00 -10.00   0.50   0.05     NA
```

```r
z[4] <- 5i
z # double -> complex
```

```
## [1]   1.0+0i -10.0+0i   0.5+0i   0.0+5i       NA
```

```r
z[1] <- "-2"
z # complex -> character
```

```
## [1] "-2"     "-10+0i" "0.5+0i" "0+5i"   NA
```

```r
z[5] <- 20
z # NA -> 20 (numeric) -> "20" (character)
```

```
## [1] "-2"     "-10+0i" "0.5+0i" "0+5i"   "20"
```

Exercises

1) Consider the following two vectors: x and y.

```r
x <- c(2, 4, 6, 8, 10)
y <- c("a", "e", "i", "o", "u")
```

What is the output of the following R commands? (BTW: they are all valid commands). Try to answer these parts without running the code in R.

a) `y[x/x]`

b) `y[!(x > 5)]`

c) `y[x < 10 & x != 2]`

d) `y[x[-4][2]]`

e) `y[as.logical(x)]`

f) `y[6 - (x/2)]`

```r
x <- c(2, 4, 6, 7, 10)
y <- c("a", "e", "i", "o", "u")

# a
y[x/x]
```

```
## [1] "a" "a" "a" "a" "a"
```

```r
# b
y[!(x > 5)]
```

```
## [1] "a" "e"
```

```r
# c
y[x < 10 & x != 2]
```

```
## [1] "e" "i" "o"
```

```r
# d
y[x[-4][2]]
```

```
## [1] "o"
```

```r
# e
y[as.logical(x)]
```

```
## [1] "a" "e" "i" "o" "u"
```

```r
# f
y[6 - (x/2)]
```

```
## [1] "u" "o" "i" "e" "a"
```

2) Consider the following R code:

```r
# peanut butter jelly sandwich
peanut <- TRUE
peanut[2] <- FALSE
yummy <- mean(peanut)
butter <- peanut + 1L
jelly <- tolower("JELLY")
sandwich <- c(peanut, butter, jelly)
```

What is the output of the following commands? Try to answer these parts without running the code in R.

a) `"jelly" != jelly`

b) `peanut & butter`

c) `typeof(yummy[peanut])`

d) `sandwich[2]`

e) `peanut[butter]`

f) `peanut %in% peanut`

g) `typeof(!yummy)`

h) `length(list(peanut, butter, as.factor(jelly)))`

```r
peanut <- TRUE
peanut[2] <- FALSE
yummy <- mean(peanut)
butter <- peanut + 1L
jelly <- tolower("JELLY")
sandwich <- c(peanut, butter, jelly)

# a
"jelly" != jelly
```

```
## [1] FALSE
```

```r
# b
peanut & butter
```

```
## [1]  TRUE FALSE
```

```r
# c
typeof(yummy[peanut])
```

```
## [1] "double"
```

```r
# d
sandwich[2]
```

```
## [1] "FALSE"
```

```r
# e
peanut[butter]
```

```
## [1] FALSE  TRUE
```

```r
# f
peanut %in% peanut
```

```
## [1] TRUE TRUE
```

```r
# g
typeof(!yummy)
```

```
## [1] "logical"
```

```r
# h
length(list(peanut, butter, as.factor(jelly)))
```

```
## [1] 3
```

3) Consider the following two vectors: x and y.

```
x <- c(1, 2, 3, 4, 5)
y <- c("a", "b", "c", "d", "e")
```

Match the following commands with their corresponding output. Try to answer these parts without running the code in R.

```
a)  y[x == 1]              ___  "a" "b" "c" "d" "e"

b)  y[x]                   ___  "e"

c)  y[x < 3]               ___  character(0)

d)  y[x/x]                 ___  "d"

e)  y[x[5]]                ___  "c" "d" "e"

f)  y['b']                 ___  NA

g)  y[0]                   ___  "a" "b"

h)  y[!(x < 3)]            ___  "c"

i)  y[x[-2][3]]            ___  "a"

j)  y[x[x[3]]]             ___  "a" "a" "a" "a" "a"
```

```
x <- c(1, 2, 3, 4, 5)
y <- c("a", "b", "c", "d", "e")

# a
y[x == 1]
```

```
## [1] "a"
```

```
# b
y[x]
```

```
## [1] "a" "b" "c" "d" "e"
```

```
# c
y[x < 3]
```

```
## [1] "a" "b"
```

```
# d
y[x/x]
```

```
## [1] "a" "a" "a" "a" "a"
```

```r
# e
y[x[5]]
```

```
## [1] "e"
```

```r
# f
y['b']
```

```
## [1] NA
```

```r
# g
y[0] # a character vector of length 0
```

```
## character(0)
```

```r
# h
y[!(x < 3)]
```

```
## [1] "c" "d" "e"
```

```r
# i
y[x[-2][3]]
```

```
## [1] "d"
```

```r
# j
y[x[x[3]]]
```

```
## [1] "c"
```

4) Which command will fail to return the first five elements of a vector x? (assume x has more than 5 elements).

a) `x[1:5]`

b) `x[c(1,2,3,4,5)]`

c) `head(x, n = 5)`

d) `x[seq(1, 5)]`

e) `x(1:5)`

```r
# a
x[1:5]
```

```
## [1] 1 2 3 4 5
```

```r
# b
x[c(1, 2, 3, 4, 5)]
```

```
## [1] 1 2 3 4 5
```

```r
# c
head(x, n = 5)
```

```
## [1] 1 2 3 4 5
```

```r
# d
x[seq(1, 5)]
```

```
## [1] 1 2 3 4 5
```

```r
# e (answer)
# x(1:5) surrounding by () means applying a function
```

5) Explain the concept of atomic structures in R.

It is one type of the most fundamental data structure in R that can contain only one type of data. An atomic vector is either logical, integer, numeric, complex, character or raw and can have any attributes except a dimension attribute (like matrices). I.e., a `factor` is an atomic vector, but a `matrix` or `NULL` are not. In short, this is basically equivalent to `is.atomic(x) && !is.null(x) && is.null(dim(x))`. To check if it is an atomic vector, use `is.vector()`.

6) Explain the concept of vectorization a.k.a. vectorized operations.

Operations occur in parallel in certain R objects (i.e. They are computed element-by-element). This allows us to write code efficiently, concisely, and easily to read than in non-vectorized languages.