2. Data types and Data structures in R

Principles of Data Science with R

Dr. Uma Ravat

PSTAT 10

Announcement:

1. Post on **Ed** https://edstem.org/ with correct tags or categories and answer questions from peers

- **Today is Week 1, Lecture 02**

See: https://tinyurl.com/AskingLectureQuestionOnEd

2. For any issue, including worksheets, contact your TA first
    - **Do not email me(instructor)** your worksheets, hw etc

- While contacting your TA via private channel on ED:
    - Start the title with @your_TAname or @HeadTA
    - select your TAs name as a category

Announcement

1. Worksheet 1 submission has been extended to Friday Sept 30 at 8am.
2. Worksheet 2 is still due 30 mins after section
3. Quiz 1 will open tomorrow(Friday) at 9am and will close at 9pm
   - Choose a 30 minute window to take it.
   - **No collaboration**
   - **No extensions**
   - Read directions carefully
   - Material covered in Lectures 00, 01, 02 : syllabus and course policies, lecture material
   - Good Luck!!

R essentials: summary

- Console and Environment Panes, Command Prompt
- Objects
  - Variables: nouns
  - Functions: verbs
  - Naming conventions
- Packages: ready made functions and datasets from others
  - Install once
  - Load every time you need it
- Help : ?
- Assignment Operator : <-
  - printing objects
- Comments: #
  - **use them!** for yourself, the grader
- Coding style : **have one** and be consistent
  - See chapters 1-3 of the tidyverse style guide
- Environment

Next we will see...

- **Data types**: What different types of data does R support?
- **Data structures**
  - What are they and why should you care?
  - What various data structures does R support?
  - How can data be accessed within the various data structures?
  - What functions are available for working with each data structure?

Q1: What types of data does `R` support?

- **character** (also known as string): `"a"`, `"PSTAT"`
- **double** (also known as **numeric**): `2`, `15.5`
  - for real or decimal numbers
- **integer** (whole numbers): `2L`
  - the `L` tells R to store the number 2 as an integer
- **logical**: `TRUE`, `FALSE` (same as 1 or 0)

Check the object's data type : `typeof()` function

```r
# Example
student <- "Ally"
typeof(student)
```

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

```r
y <- 1
typeof(y)
```

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

```r
z <- as.integer(y) # OR use L notation while creating
typeof(z)
```

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

```r
pass <- TRUE
typeof(pass)
```

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

"What does the word data structure mean?"

A *data structure* is a mechanism to group related data values into an **object.**

Remember what *John Chambers* said

> *Everything that exists in R is an object.*

Now that you know a *data structure* groups data into objects.

WAP: With a Peer! Discuss what would you need to know about data structures in R to be a successful *R programmer* or *data scientist*?

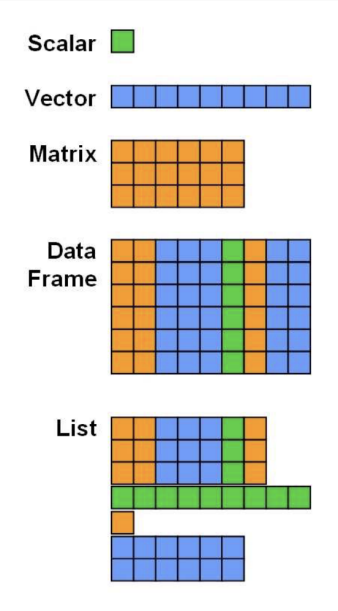
Need to know

- What are the possible data structures? Is there just one or many that 'R supports?
- What are the differences between different data structures?
- How do you create each of them in `R`?
- How to use them to store and access data?
- What functionality (functions) does each come with?

Analogy : law of the land

R has many data structures :

- **scalar:** stores one value at a time
- **(atomic) vector :** stores a sequence of values, all of the same type
- **matrix :** all of same type, data is stored in rows and columns
- **list :** elements can have different types
- **data frame :**
- **factors :**



SCALAR data structure

Scalars can hold only one value at a time.

EXAMPLE

```
(x <- 4) # no need to write print, use () instead!
```

```
## [1] 4
```

```
(y <- "Hello! Have you fallen asleep?")
```

```
## [1] "Hello! Have you fallen asleep?"
```

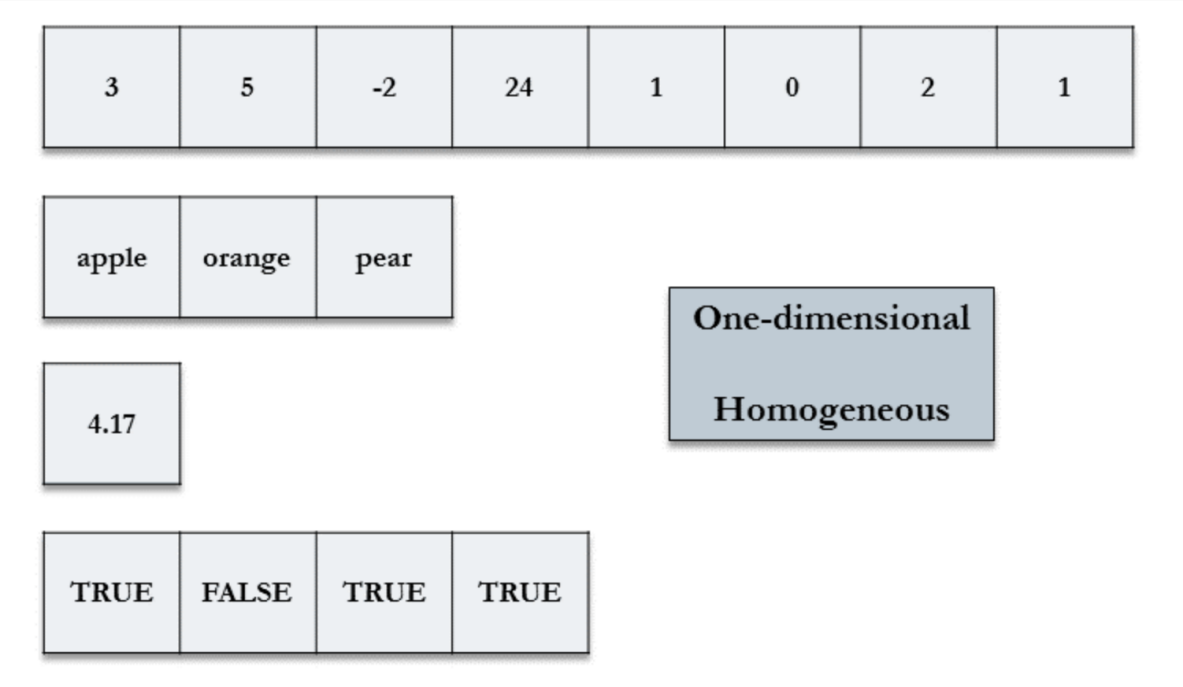
```
(asleep <- FALSE)
```

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

Vectors

Vectors store a sequence of values, all of the same type

- most common and basic data structure in `R`
- workhorse of `R`
- also refered to as **atomic vectors**
- one-dimensional and homogeneous data structure
- A scalar data structure is just a vector of length 1.



Let's look at creating vectors and some functions

- c() : the `combine` function

Functions:

- typeof() : What type of data is stored in the vector
- length() : the number of elements contained in the vector.
- sort() : the `sort` or `ordering` function

EXAMPLES OF VECTORS and R functions for vectors

```r
# Numeric vector
a <- c(1, 6, 5.3, 6, -2, 4) #as opposed to a<-c(1,6,5.3,6,
typeof(a)
```

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

```r
#Character vector
students <- c("Jaden","Diana","Daisy", "Maribel", "Tera")
typeof(students)
```

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

```r
#Logical vector
have_met_OH <- c(FALSE, TRUE, FALSE, TRUE, FALSE)
typeof(have_met_OH)
```

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

length()

```
# How long are each of these vectors?
length(a)
```

```
## [1] 6
```

```
length(students)
```

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

```
length(have_met_OH)
```

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

sort()

```
# How does R sort these vectors
# What's the default sort order, how do you change it? Try
# Check ?sort if you haven't already
sort(a) # default is increasing.
```

```
## [1] -2.0  1.0  4.0  5.3  6.0  6.0
```

```
sort(students)
```

```
## [1] "Daisy"   "Diana"   "Jaden"   "Maribel" "Tera"
```

```
sort(have_met_OH) # 0 vs 1
```

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

```r
# After sorting a remains unsorted,
a <- c(1, 6, 5.3, 6, -2, 4)
sort(a)
```

```
## [1] -2.0  1.0  4.0  5.3  6.0  6.0
```

```r
a
```

```
## [1]  1.0  6.0  5.3  6.0 -2.0  4.0
```

```r
#the sorted vector must be saved to get it in sorted order
( a <- sort(a) )
```

```
## [1] -2.0  1.0  4.0  5.3  6.0  6.0
```

Coercing the data type of a vector

```
x <- c(1, 2, 3)
typeof(x)
```

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

To **explicitly** create integers: use L

```
x1 <- c(1L, 2L, 3L)
typeof(x1)
```

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

OR *coerce* to the integer type using `as.integer()`

```
x2 <- as.integer(x)
typeof(x2)
```

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

R allows you to coerce to any data type

Know what you are doing with `as.datatype()` functions

```
x
```

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

```
typeof(x)
```

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

```
(x3 <- as.character(x))
```

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

```
typeof(x3)
```

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

Some more functions for vectors

```
(score <- c(10, 0, 5))
```

```
## [1] 10  0  5
```

```
diff(score)
```

```
## [1] -10   5
```

```r
# Assign names to entries in our score vector
names(score) <- c("Ally", "Maribel", "Chase")
score
```

```
##    Ally Maribel   Chase
##      10       0       5
```

```r
# view the names that we assigned to our score vector.
names(score)
```

```
## [1] "Ally"    "Maribel" "Chase"
```

```r
attributes(score) # metadata about the object
```

```
## $names
## [1] "Ally"    "Maribel" "Chase"
```

Automatic coercion of vectors

- Vectors can only contain one data type.

If you try to use the c() function to create a vector with elements of different type:

- R determines a common vector type.
- This is called COERCION.
- Vectors are coerced to the simplest type required to represent all information.
- **For example:** vectors containing numeric elements AND character elements are **coerced** to a character vector.

Automatic coercion in R:

Be careful, R doesn't complain while coercing

This is a source of frustration for beginning programmers!

```
a <- c(1, 2, 3); b <- c("Bob", "5")
cat(paste("Type of a is : ", typeof(a)),
    paste("Type of b is :", typeof(b)),sep='\n')
```

```
## Type of a is :  double
## Type of b is : character
```

```r
a <- c(1, 2, 3); b <- c("Bob", "5")
# All elements will be coerced into the simplest type, cha
(x <- c(a, b)) # c() can combine two vectors
```

```
## [1] "1"   "2"   "3"   "Bob" "5"
```

```r
typeof(x)
```

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

```r
# check that x is indeed a vector
is.vector(x)
```

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

More (Faster) ways to create (long) vectors

- `:` - the colon operator
- `seq()` : the `sequence generation` function
- `rep()` : the `replicate` function

Creating a vector

Create a vector with elements: 1,2,3,4,5,6,7,8,9,10

```
# 1. Using c
x <- c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
x
```

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

```
typeof(x)
```

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

```
is.vector(x)
```

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

2. Using :

Create a vector with elements: 1,2,3,4,5,6,7,8,9,10

```
x <- 1:10
x
```

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

```
typeof(x)
```

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

```
is.vector(x)
```

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

3. Using the seq function

Construct a vector with elements: 3.0 3.8 4.6 5.4 6.2 7.0

```
(x <- seq(from=3, to=7, by=0.8))
```

```
## [1] 3.0 3.8 4.6 5.4 6.2 7.0
```

```
typeof(x)
```

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

```
is.vector(x)
```

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

SYNTAX: seq()

seq(from, to)

seq(from, to, by = , length =)

4. Using the replication function, rep()

`rep()` function creates a vector of repeated values.

- Create a vector with elements: 1 1 1 2 2 2 3 3 3

```
x <- rep(1:3, each = 3)
typeof(x)
```

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

```
is.vector(x)
```

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

Vector math:

```
(a <- 1:5); (b <- 6:10)
```

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

```
## [1]  6  7  8  9 10
```

```
a + b # try other math operations
```

```
## [1]  7  9 11 13 15
```

Many operations in R are already vectorized

```
(x <- (5:10)^2)
```

```
## [1]  25  36  49  64  81 100
```

```
log(x)
```

```
## [1] 3.218876 3.583519 3.891820 4.158883 4.394449 4.60517
```

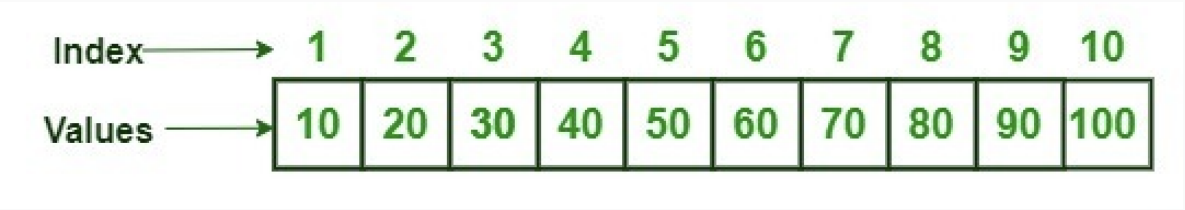
Vector math: Adding a scalar to a vector!

R adds the scalar to each element!

```
x <- 1:10
x + 6
```

```
##  [1]  7  8  9 10 11 12 13 14 15 16
```

How do we access elements of a vector?



- Access elements using their index.
- Accessing certain elements is also called **subseting** a vector

Accessing elements of a vector 1. Using the square bracket operator [ ]

```r
# (a) Construct a vector x with elements 1,7,3,10,5.
x <- c(1, 7, 3, 10, 5)
```

```r
# b) Write code to return the 4th element of x
# (index is 4)
x[4]
```

```
## [1] 10
```

Subsetting a vector: using [ ]

```
# c) Write code to return every element of x
# except the 2nd (index is not 2)
x[-2]
```

```
## [1]  1  3 10  5
```

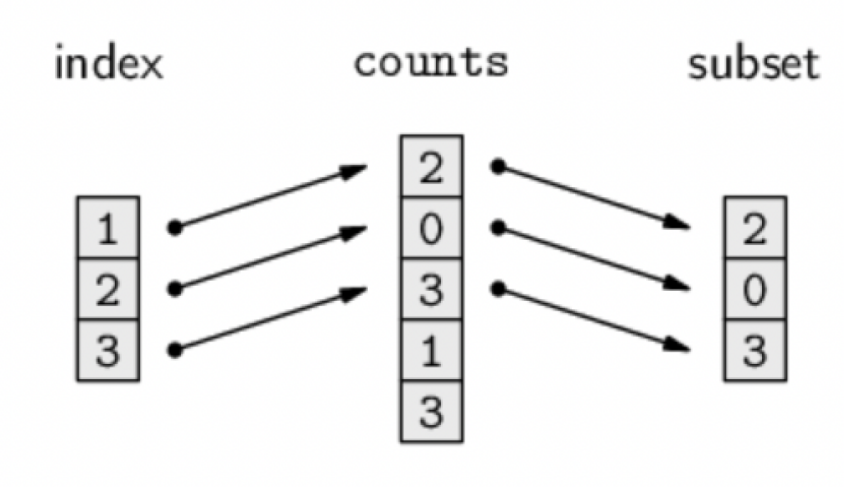
```
# (d) Assign the value 100 to the 4th element of x
x[4] <- 100
x
```

```
## [1]   1   7   3 100   5
```

Subsetting a vector: using : operator

for extracting *successive elements* of a vector

Construct a vector named counts with elements 2, 0, 3, 1, 3 and extract the subset containing the elements 2, 0, 3.




```
counts <- c(2, 0, 3, 1, 3)
counts[ 1:3 ]
```

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

Subsetting a vector: Using `c()` function

Extract non-successive elements using the combine c() function

```
counts <- c(2, 0, 3, 1, 3)
# return the 1st and 3rd element of counts.
counts[ c(1,3) ]
```

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

Subsetting a vector: via selection criteria

Extract all elements of a vector that match a selection criteria by comparision:

- $<$ for less than
- $>$ for greater than
- $\leq$ for greater than or equal to
- $\geq$ for less than or equal to
- $==$ for equal to each other
- $!=$ for not equal to each other

```r
price  <- c(500, -10, 30, 20, -10, -3 , - 2, 100, 90  )
# is a value positive?
positive_price <- price > 0
positive_price
```

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

```r
# select all positive values
price[price > 0]
```

```
## [1] 500  30  20 100  90
```

In Data science, selecting values that satisfy a property is called **filtering**. We **filtered** positive prices from all price values.

WAP: With a Peer! What do you think `sum`, `max`, `min`, `which.max`, `which.min` functions would do when applied to a vector? Try it out!

Summary: Post-Lecture 2 To DO

1. Review the lecture again
2. On this slide/paper write down a summary of today's lecture. Include all functions we went over and a short description of what each function does.

You will be asked to attach this to your homework.

1. What's wrong with this code? (`esc` will rescue you!_)

```
hello <- "Hello world!
```

Suppose we have test scores for 5 students: Bob, Alice, Alex, Juan and Amy.

Their scores are 8, 7, 8, 10, and 5 respectively.

1. Create a vector of these scores.
2. Find the mean score in two ways (using `mean` and using `sum`).
3. Find the median score.
4. Assign the name of each student to their test score.
5. Retrieve Alice's score in two ways.
6. Retrieve Amy's and Alice's score, in that order.
7. Retrieve all except Amy's score.

questions you should be able to answer

- "What are the different data types in R?"
- "What are the different data structures in R?"
- "How do I access data within the various data structures?"