

# INTRODUCTION TO DATA SCIENCE

## Lecture 3

Dr. Ibrahim Radwan

# OUTLINE

---

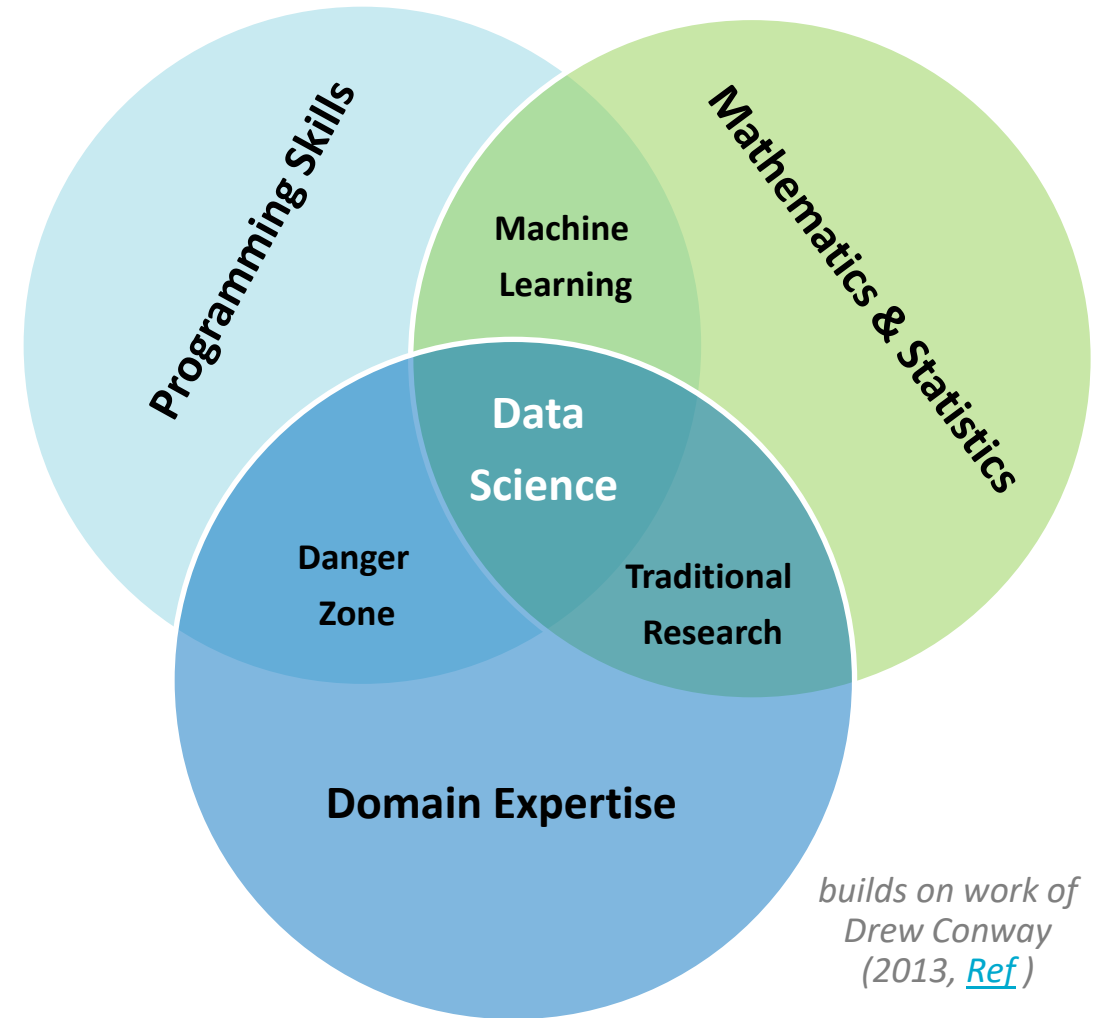
- Recap from the last two lectures
- Data structures in R
- Vectors in R
- Operation on vectors
- Conditional statements
- Loop statements

# DATA SCIENCE

- What is the Data Science?
- Why is Data Science so important?

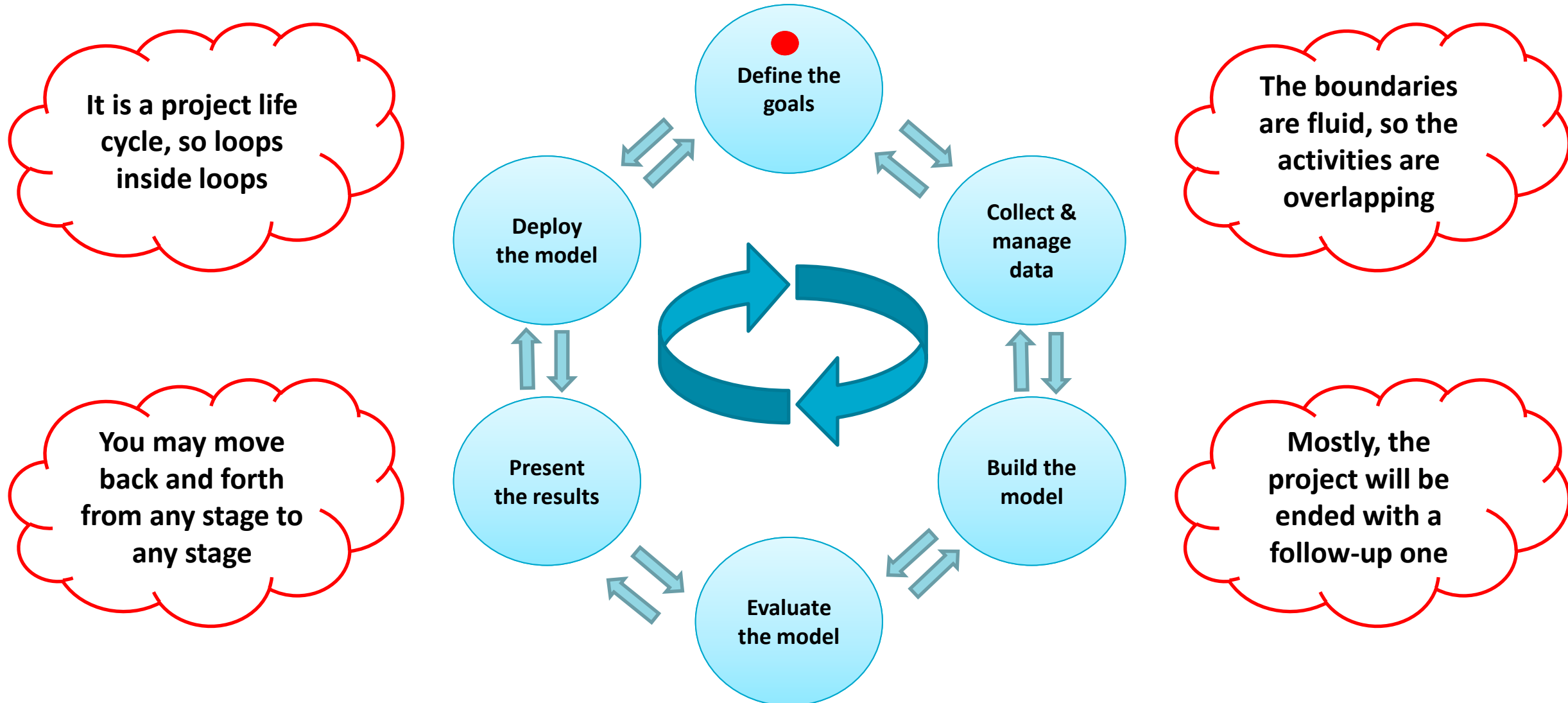
Data

Information



Data Science Venn diagram

# STAGES OF A DATA SCIENCE PROJECT



- Data Science code of ethics should involve the principles and values that govern our behaviour and actions with the data toward the individuals and the community.
- Data science is meant to extract good things from the data, so it is all about doing what is good for the people and what is better for lives.
- It is a community effort, so better to engage in a public conversation regularly about code of ethics.
- Always ensure that valid and correctly interpreted results are provided from your analysis.
- Don't go with the data beyond what is specified in the project specifications

## Basic Data Types

Character

“AA”, “bb”

Numeric

4.14, 2.65

Integer

7, 12

Logical

TRUE, FALSE

Complex

1+7i, 8+2i

## Variables

```
X <- 23
```

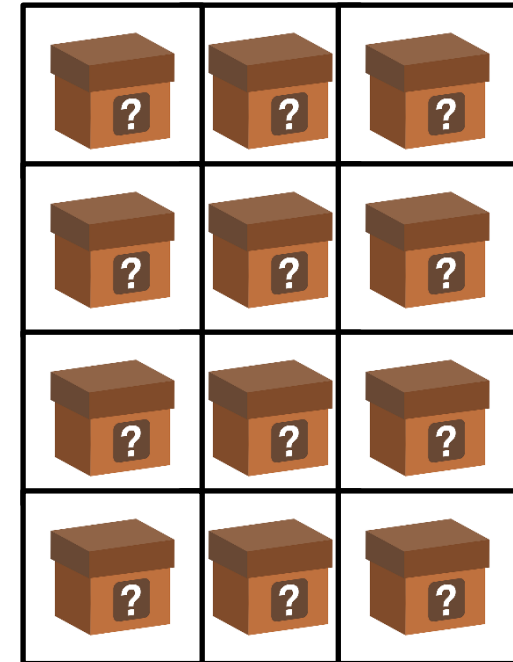
naming conventions

## Operators

Arithmetic

Logical

- Collection of elements grouped under one name
  - e.g. container of boxes
    - What type of data to put in?
    - How to access these elements?
    - How to perform operations on these elements?
  - To answer these questions, the data structures can be organized into two categories:
    1. With **similar type** of elements
    2. With **dissimilar type** of elements



# DATA STRUCTURES (2)

## Similar type of items

### Homogeneous data structures

Atomic vector      1-D

Matrix              2-D

Array                n-D

## Dissimilar type of items

### Heterogeneous data structures

List                  1-D

Data frame        2-D

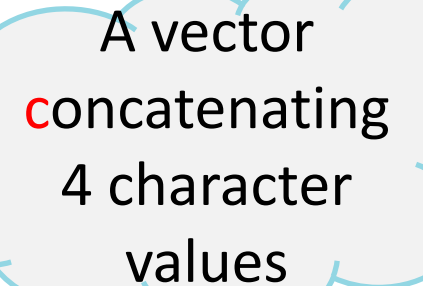


# VECTOR

- Homogeneous data types
- Data are arranged in one dimension
- How to construct/create a vector?



- `my.vec <- c("elem1", "elem2", "elem3", "elem4")`
- To get the number of elements in a vector, we use, **length** function
  - `length(my.vec)` # result in 4, as the number of elements in the vector
- Type of the elements in a vector can be extracted using:
  - `mode(my.vec)` or `class(my.vec)`
- To access elements in a vector, we use square [] and the index/ indices of the elements.
  - `x <- my.vec[2]` to access the value of the second element, or
  - `x <- my.vec[2:4]` to access the values of second, third and fourth elements



A vector  
concatenating  
4 character  
values

# VECTOR (2)

## Code Example

```
1 # Create atomic vectors
2
3 # Character
4 houses.addresses <- c("7 George st", "18/5 Irwan crescent", "8 Morad close", "1/2 London Circuit")
5 houses.addresses
6 length(houses.addresses)
7 is.character(houses.addresses)
8 str(houses.addresses)
9
10 # Numeric
11 houses.area <- c(420.5, 220.15, 750.4, 120.5)
12 houses.area
13 length(houses.area)
14 str(houses.area)
15
16 # Integer
17 houses.bedrooms <- c(4L, 3L, 5L, 2L)
18 houses.bedrooms
19 length(houses.bedrooms)
20 str(houses.bedrooms)
21
22 # Logical
23 houses.has.garden <- c(TRUE, FALSE, TRUE, FALSE)
24 houses.has.garden
25 length(houses.has.garden)
26 str(houses.has.garden)
```

# VECTOR (3)

- What are the types of the following vectors?
  - `u <- c(4, 7, 23.5, 76.2, 80, "rrt")`
  - `v <- c(4, 6, NA, 2)`
  - `k <- c(TRUE, FALSE, FALSE, NA, TRUE)`
- Let us assume `v <- c(45, 243, 78, 343, 445, 44, 56, 77)`
  - What is the output of the following?
    - `mode(v)`
    - `length(v)`
    - `v[3]`
    - `v[2:5]`
    - `v[9]`
    - `range(v)`

# OPERATIONS ON VECTORS

---

Arithmetic & Logical  
operations

Subsetting

Coercions

Let us code with R ...

# OPERATIONS ON VECTORS (2)

Code Example

```
1 x <- 6:15
2 y <- 21:30
3 # arithmetic op on vectors
4 x+y
5 x * y
6 x / y
7 # logical op on vectors
8 x > 8
9 x >= 8
10 y == 28
11 # subsetting
12 x[1]
13 x[2]
14 y[2:5]
15 x[c(T,F,F,T,F,T,T,F, F,T)]
16 x[x >= 7]
17 # coercions: converting one type to another
18 # implicit coercions
19 z <- c(8, 7.4, 14, "5")
20 str(z)
21 # explicit coercions
22 # sensible coercions
23 as.numeric(x >= 9) # converting logical values to numeric values
24 as.character(x)
25
26 houses.area <- c(420.5, 220.15, 750.4, 120.5)
27 houses.area
28 as.integer(houses.area)
29
30 # non sensible coercions
31 houses.addresses <- c("7 George st", "18/5 Irwan cresent", "8 Morad close", "1/2 London Circuit")
32 houses.addresses
33 as.numeric(houses.addresses) # warning here and NAs are the result |
```

# OPERATIONS ON VECTORS (3)

- Combining vectors

- We also can use `c( )` to concatenate multiple vectors, e.g.

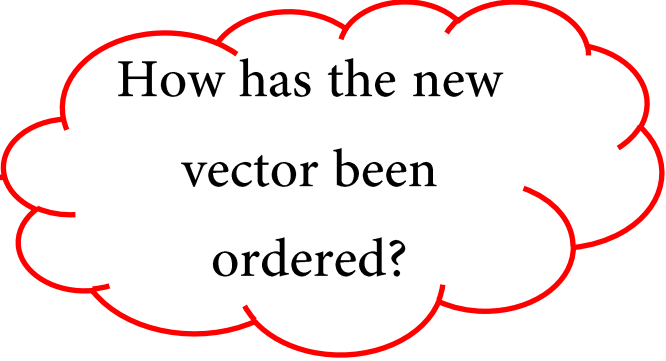
- fruits `<- c("Apple", "oranges", "banana")`

- vegetables `<- c("cabbage", "spinach", "tomatoes")`

- all\_basket\_items `<- c(fruits, vegetables)`

- `print(all_basket_items)`

- `[1] "Apple" "oranges" "banana" "cabbage" "spinach" "tomatoes"`



How has the new  
vector been  
ordered?

# VECTORIZATION

- One of the most powerful aspects of the R language
- Several functions and operations can be applied directly to produce an “*equal-sized*” vector of results.

```
> v1 <- c(4, 6, 8, 24)
> 2 * v1
[1] 8 12 16 48
> |
```

```
> v <- c(4, 7, 23.5, 76.2, 80)
> sqrt(v)
[1] 2.000000 2.645751 4.847680 8.729261 8.944272
> |
```

```
> v1 <- c(4, 6, 8, 24)
> v2 <- c(10, 2, 4)
> v1 + v2
[1] 14 8 12 34
Warning message:
In v1 + v2 :
  longer object length is not a multiple of shorter object length
> |
```

- R allows us to generate increasing and decreasing ordered vectors of integers or real numbers, *e.g.*

```
> x <- 1:100
> x
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21
[22] 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42
[43] 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63
[64] 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84
[85] 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
> y <- -12:0
> y
 [1] -12 -11 -10 -9 -8 -7 -6 -5 -4 -3 -2 -1  0
> |
```

- For real numbers, you can use **seq( )** function, *e.g.*

```
> seq(1, 4, 0.5)
 [1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0
> seq(from= -4, to= 0, length=4)
 [1] -4.000000 -2.666667 -1.333333  0.000000
> seq(length= 10, from= -2, by=0.2)
 [1] -2.0 -1.8 -1.6 -1.4 -1.2 -1.0 -0.8 -0.6 -0.4 -0.2
> |
```



- R also allows us to generate *random* sequences using:

*rfunc(n, par1, par2, ...)*, where “*func*” is the name of the probability distribution, *n* is the number of data to generate and *par1*, *par2*, ... are the parameters of the distribution. For example:

- Generate 10 samples from normal distribution:

```
> rnorm(10)
[1]  0.6913404  0.8795089  0.8486875  2.4803172  1.0548602 -0.4222548  0.4652331
[8]  0.6359163 -0.7445954 -0.5483190
> rnorm(10, mean=4, sd=3)
[1] -1.7364438  9.4358829  3.9433952  0.8107398  3.2227736  7.1686271  8.6734043
[8]  5.4138813 10.4945642  7.3487464
> |
```

- Generate 5 samples from Student t distribution

```
> rt(5, df=10)
[1] -1.3324269 -1.1507256 -0.3581286  0.9027296  1.1603391
> |
```

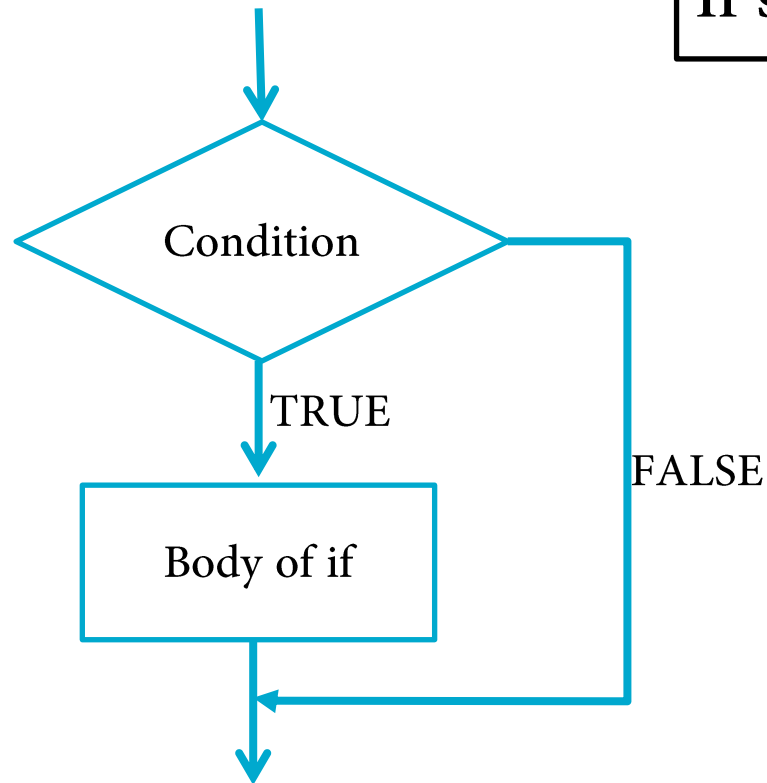
# REPEATING VECTORS

- To generate sequences with certain pattern, you can use *rep()* function, e.g.

```
> vec <- rep(1:2, 5)
> vec
[1] 1 2 1 2 1 2 1 2 1 2
> vec <- rep(1:2, each=3)
> vec
[1] 1 1 1 2 2 2
> vec <- rep(1:2, length.out=7)
> vec
[1] 1 2 1 2 1 2 1
> vec <- rep(c("m", "f"), 3)
> vec
[1] "m" "f" "m" "f" "m" "f"
> |
```

# CONDITIONAL STATEMENTS

## If statement



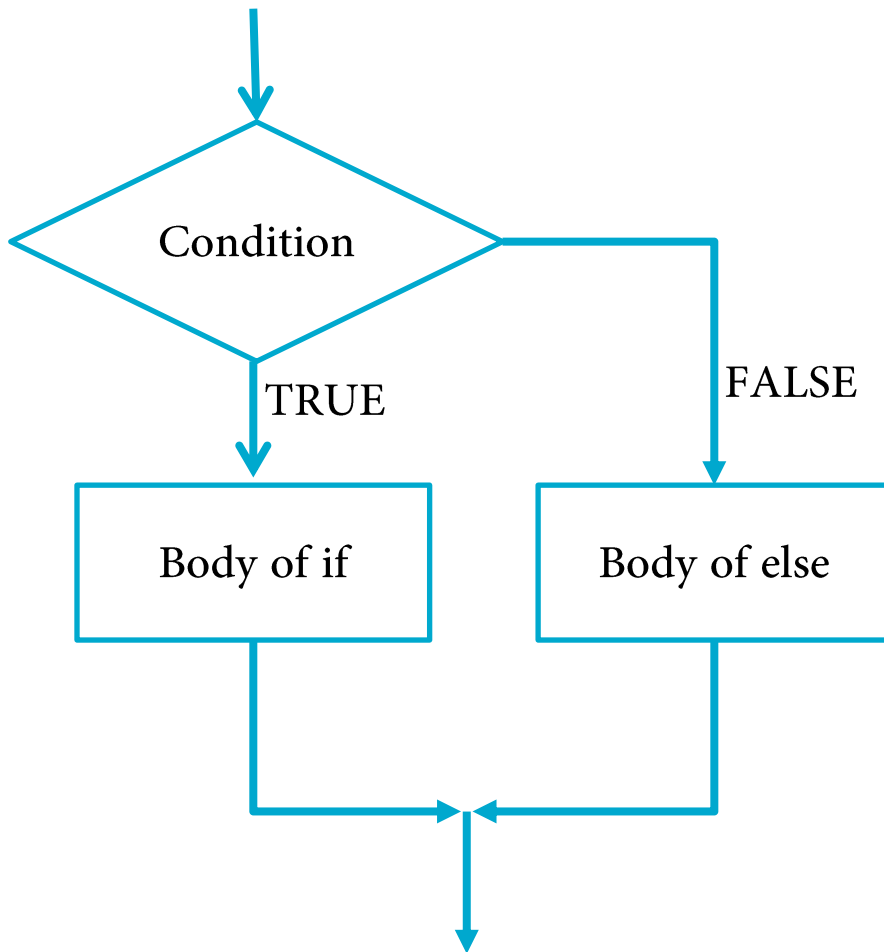
```
if (Condition) {  
    # block of statements to be executed if  
    the condition is evaluated to TRUE  
}
```

# CONDITIONAL STATEMENTS (2)

## (if) Example

```
1 # Generate random marks for the students
2 students.length <- 50
3 students.marks <- rnorm(students.length, mean= 50, sd=50)
4
5 # compute average
6 students.average <- mean(students.marks)
7 print(paste("average mark of the class is:", students.average))
8
9 # check average mark of the class
10 if(students.average >= 50){
11     print(paste("The performance is OK, as the average =", students.average))
12 }
13
14 print("Test is completed!")
```

# CONDITIONAL STATEMENTS (3)



## ► If ... else statement

```
if (condition){
```

```
    # block1 of statements to be executed  
    if the condition is evaluated to TRUE
```

```
}else{
```

```
    # block2 of statements to be executed  
    if the condition is evaluated to FALSE
```

```
}
```

# CONDITIONAL STATEMENTS (4)

## (if else) Example

```
1 # Generate random marks for the students
2 students.length <- 50
3 students.marks <- rnorm(students.length, mean= 50, sd=50)
4
5 # compute average
6 students.average <- mean(students.marks)
7 print(paste("average mark of the class is:", students.average))
8 |
9 # check average mark of the class
10 if(students.average >= 50){
11     print(paste("The performance is OK, as the average =", students.average))
12 }else{
13     print(paste("The performance is bad, as the average =", students.average))
14 }
15
16 print("Test is completed!")
```

# CONDITIONAL STATEMENTS (5)

## Multiple (if else) example

```
1 # Generate random marks for the students
2 students.length <- 50
3 students.marks <- rnorm(students.length, mean= 50, sd=50)
4
5 # compute average
6 students.average <- mean(students.marks)
7 students.marks <- students.marks + 10
8 print(paste("average mark of the class is:", students.average))
9
10 # multiple if else
11 if(students.average < 50){
12     print(paste("The performance is bad, as the average =", students.average))
13 }else if(students.average < 60){
14     print(paste("The performance is Ok, as the average =", students.average))
15 }else{
16     print(paste("The performance is excellent, as the average =", students.average))
17 }
18
19 print("Test is completed!")
20
```

# CONDITIONAL STATEMENTS (6)

- Can we apply (if... else) on a vector?
  - What is the result of running this code?

- `x <- c(0, 1, 2, -4, 5)`

- `if(x > 0){ print(x) }` ←

Warning message:

the condition has length > 1 and only the first element will be used.

- The solution is by using “ifelse” as following:

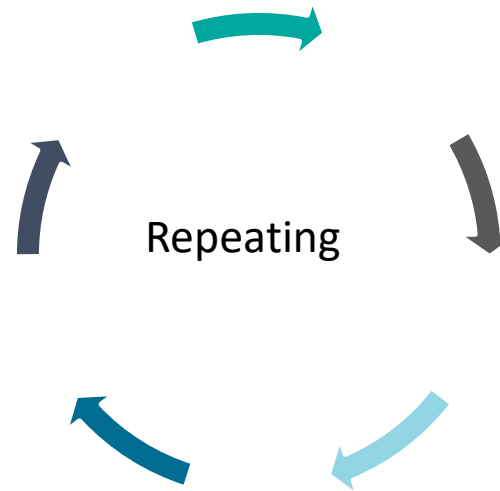
```
ifelse(Boolean_expresson, statementA, statementB)
```

```
> x <- c(0, 1, 2, -4, 5)
> result = ifelse(x > 0, 1/x, NA)
> result
[1] NA 1.0 0.5 NA 0.2
> |
```



# LOOP STATEMENTS

- Repeat the execution of some statements for number of times
- Three ways to do that:
  - for loop
  - while loop
  - repeat loop

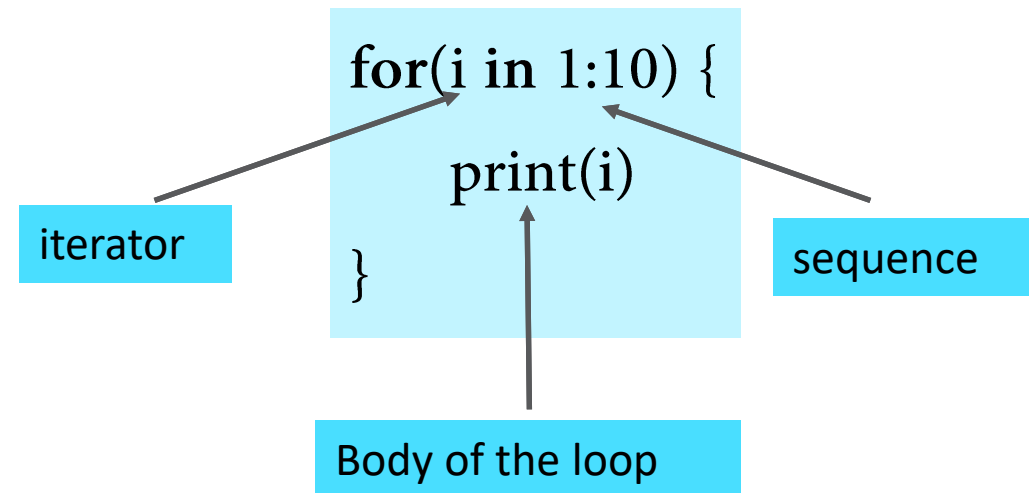


# FOR LOOP STATEMENT

- For loops are pretty much the most usable looping construct that you will need in R
- For loops take an iterator variable and assign it successive values from a sequence, vector or a list.

```
1 page.limit <- 10
2 for(count in 1:page.limit){
3   print(paste("page number is:", count))
4 }
5 print("Flipping the pages has finished")
6 |
```

Code Example



# WHILE LOOP STATEMENT

- While loops begin by testing a condition, if it is true, then they execute the loop body
- Once the loop body is executed, the condition is tested again, and so forth, until the condition is false, after which the loop exits.

```
1 page.limit <- 10
2 count <- 0
3 while(count < page.limit){
4   count <- count + 1
5   print(paste("page number is:", count))
6 }
7 print("Flipping the pages has finished")
8
```

Code Example

Condition

Initialize the  
iterator

Body of the loop

Increment/decrement

# KEY TAKEAWAYS

---

- Data structures are collection of elements where homogenous or heterogeneous data types can be stored in.
- Vector is the most common data structure in R
- Vectorized operations make R quite powerful when dealing with data analysis
- Sequences are kind of vectors where their elements are put in order
- Conditional statements are useful in specifying what to be executed based on whatever conditions
- Loop statements provide more flexibility by allowing us to execute statements number of times

# RECOMMENDED READING

---

- You are recommended to read chapters 20 and 21 from “R for Data Science” book within these links:
  - <https://r4ds.had.co.nz/vectors.html>
  - <https://r4ds.had.co.nz/iteration.html>
- Also, you may check some relevant sections (e.g., 3.7, 3.8, 3.12, 4.1 and 4.4) in “Introduction to Data Science, Data Analysis and Prediction Algorithms with R” book by Rafael A. Irizarry. The book is published online within this link:
  - <https://rafalab.github.io/dsbook/>