# Data Structure

Antonio Vega

# Vectors

# Data types in R

A **vector** is an ordered sequence of data. `R` has many types of data, for example:

- `logical`: logical (`TRUE` or `FALSE`)
- `integer`: whole numbers, $\mathbb{Z}$
- `numeric`: real numbers, $\mathbb{R}$
- `complex`: complex numbers, $\mathbb{C}$
- `character`: words

In the vectors of `R`, all their objects must be of the same type: all numbers, all words, etc. When we want to use vectors formed by objects of different types, we will have to use **generalized lists**, 'lists' that we will see at the end of the topic.

# Basic

- `c()`: to define a vector
- `scan()`: to define a vector
- `fix(x)`: to visually modify the vector $x$
- `rep(a, n)`: to define a constant vector that contains the data $a$ repeated $n$ times

```
c(1,2,3)
```

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

```
rep("Tono",5)
```

```
[1] "Tono" "Tono" "Tono" "Tono" "Tono"
```

# scan() function

This functions also work with URL with text files

**Example**

This is an example of how to create a vector that contains 3 copies of 1 9 9 8 0 7 2 6 with the scan function:

```
> scan()
1: 1 9 9 8 0 7 2 6
9: 1 9 9 8 0 7 2 6
17: 1 9 9 8 0 7 2 6
25:
Read 24 items
 [1] 1 9 9 8 0 7 2 6 1 9 9 8 0 7 2 6 1 9 9 8 0 7 2 6
>
```

# Progressions and Sequences

An arithmetic progression is a succession of numbers such that the **difference**, $d$, of any pair of successive terms in the sequence is constant.

$$a_n = a_1 + (n-1)\ cdotd$$

- `seq (a, b, by=d)`: to generate an arithmetic progression of difference $d$ that starts in $to$ up get to $b$
- `seq (a, b, length.out=n)`: define arithmetic progression of length $n$ ranging from $a$ to $b$ with difference $d$. Therefore $d = (b-a)/(n-1)$
- `seq (a, by= d, length.out=n)`: define the arithmetic progression of length $n$ and difference $d$ starting in $a$
- `a:b`: define the sequence of consecutive numbers ** ($\mathbb{Z}$) between two numbers $a$ and $b$

# Example of sequence

- This is an example of a sequence 5, 8.5, 12, 15.5, 19, 22.5, 26, 29.5, 33, 36.5, 40, 43.5, 47, 50.5, 54, 57.5
- If we want a sequence of **n** lengh we can use 5, 14.1666667, 23.3333333, 32.5, 41.6666667, 50.8333333, 60
- If we want to start in n number and have a determinate lenght with n step we use 5, 8, 11, 14, 17, 20, 23

# Sequences Exercises

- Numbers sequence from 1 ot 20: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20
- First 20 pair numbers: 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40
- 30 equidistant numbers bettwen 17 and 98 with 4 significative decimals: [1] 17.0000 19.7931 22.5862 25.3793 28.1724 30.9655 33.7586 36.5517 [9] 39.3448 42.1379 44.9310 47.7241 50.5172 53.3103 56.1034 58.8966 [17] 61.6897 64.4828 67.2759 70.0690 72.8621 75.6552 78.4483 81.2414 [25] 84.0345 86.8276 89.6207 92.4138 95.2069 98.0000

# Example of vector

This is another example to generate a vector, c is also named as concatenation:

```
c(rep(pi, 5), 5:10, -7) -> x
x
```

```
 [1]  3.141593  3.141593  3.141593  3.141593  3.141593  5.0
 [8]  7.000000  8.000000  9.000000 10.000000 -7.000000
```

```
c(0, x, 10, x, 20)
```

```
 [1]  0.000000  3.141593  3.141593  3.141593  3.141593  3.1
 [8]  6.000000  7.000000  8.000000  9.000000 10.000000 -7.0
[15]  3.141593  3.141593  3.141593  3.141593  3.141593  5.0
[22]  7.000000  8.000000  9.000000 10.000000 -7.000000 20.0
```

# Functions

When we want to apply a function to each of the elements of a data vector, the `sapply` function saves us having to program with loops in `R`:

- ''sapply (vector_name, FUN = function_name)'': to apply this function to all vector elements
- `sqrt(x)`: Calculate a new vector with the square roots of each of the elements of the vector $x$

# Example of use sapply function

```
x <- 1:10
sapply(x, FUN = function(w){sqrt(w)})
```

```
 [1] 1.000000 1.414214 1.732051 2.000000 2.236068 2.449490
 [8] 2.828427 3.000000 3.162278
```

```
sapply(x, function(w)sqrt(w))
```

```
 [1] 1.000000 1.414214 1.732051 2.000000 2.236068 2.449490
 [8] 2.828427 3.000000 3.162278
```

# Functions

Given a vector of data $x$ we can calculate many statistical measures about it:

- `length(x)`: Calculate the length of the vector $x$
- `max(x)`: calculates the maximum of the vector $x$
- `min(x)`: Calculate the minimum of the vector $x$
- `sum(x)`: calculates the sum of the entries of the vector $x$
- `prod(x)`: Calculate the product of the vector entries $x$

# Functions

- `mean(x)`: calculates the arithmetic mean of the entries in the vector $x$
- `diff(x)`: calculates the vector formed by the successive differences between entries of the original vector $x$
- `cumsum(x)`: calculates the vector formed by the cumulative sums of the original vector entries $x$
  - Allows to define sequences described by summation
  - Each entry of `cumsum(x)` is the sum of the $x$ entries up to its position

# Functions

```
square_fun = function(x){x^2}
v = c(1,2,3,4,5,6)
sapply(v, FUN = square_fun)
```

```
[1]  1  4  9 16 25 36
```

```
mean(v)
```

```
[1] 3.5
```

```
cumsum(v)
```

```
[1]  1  3  6 10 15 21
```

# Order

- `sort(x)`: order the vector in natural order of the objects that form it: the increasing numerical order, alphabetical order ...
- `rev(x)`: invert the order of the vector elements $x$

```
v = c(1,7,5,2,4,6,3)
sort(v)
```

```
[1] 1 2 3 4 5 6 7
```

```
rev(v)
```

```
[1] 3 6 4 2 5 7 1
```

# Order Exercises

```
v = c(1,7,5,2,4,6,3)
s = sort(v)
rev(s)
```

```
[1] 7 6 5 4 3 2 1
```

```
v = c(1,7,5,2,4,6,3)
sort(v, decreasing = TRUE)
```

```
[1] 7 6 5 4 3 2 1
```

```
v = c(1,7,5,2,4,6,3)
s = rev(v)
sort(s)
```

```
[1] 1 2 3 4 5 6 7
```

# Subvectors

- `vector[i]`: gives the $i$ -th entry of the vector
  - R indexes start at 1
  - `vector[length(vector)]`: gives us the last entry of the vector
  - `vector[a:b]`: if $a$ and $b$ are two natural numbers, the subvector gives us with the entries of the original vector that go from the $to$-th position until the $b$-th.
  - `vector [-i]`: if $i$ is a number, this subvector is made up of all the entries of the original vector except the $i$ -th entry. If $i$ turns out to be a vector, then it is an index vector and creates a new vector with the entries of the original vector, whose indexes belong to $i$
  - `vector[-x]`: if $x$ is a vector (of indices), then this is the complement of vector $[x]$

# Subvectors

- Also has logic operators:
  - ==: $=$
  - !=: $\neq$
  - >=: $\geq$
  - <=: $\leq$
  - <: $<$
  - >: $>$
  - !: NO logic
  - &: AND logic
  - |: OR logic

# Subvectors

```
v = c(14,5,6,19,32,0,8)
v[2]
```

```
[1] 5
```

```
v[-c(3,5)]
```

```
[1] 14  5 19  0  8
```

```
v[v != 19 & v>15]
```

```
[1] 32
```

# Subvectors

```
w = seq(3, 50, by = 3.5)
w
```

```
 [1]  3.0  6.5 10.0 13.5 17.0 20.5 24.0 27.5 31.0 34.5 38.0
```

```
w[4:8]
```

```
[1] 13.5 17.0 20.5 24.0 27.5
```

```
w[8:4]
```

```
[1] 27.5 24.0 20.5 17.0 13.5
```

```
w[seq(2, length(w), by = 2)]
```

```
[1]  6.5 13.5 20.5 27.5 34.5 41.5 48.5
```

# Subvectors

```
w[seq(1, length(w), by = 2)]
```

```
[1]  3 10 17 24 31 38 45
```

```
w[-seq(2, length(w), by = 2)]
```

```
[1]  3 10 17 24 31 38 45
```

```
w[(length(w)-2):length(w)]
```

```
[1] 41.5 45.0 48.5
```

```
w[c(1,5,6)]
```

```
[1]  3.0 17.0 20.5
```

# Subvectors

```
w[w>30]
```

```
[1] 31.0 34.5 38.0 41.5 45.0 48.5
```

```
w[w>20 & w<40]
```

```
[1] 20.5 24.0 27.5 31.0 34.5 38.0
```

```
w[w!=3 & w!=17]
```

```
 [1]  6.5 10.0 13.5 20.5 24.0 27.5 31.0 34.5 38.0 41.5 45.0
```

```
w[w<10|w>40]
```

```
[1]  3.0  6.5 41.5 45.0 48.5
```

# Subvectors

```
w[w>=10]
```

```
 [1] 10.0 13.5 17.0 20.5 24.0 27.5 31.0 34.5 38.0 41.5 45.0
```

```
w[w>10]
```

```
 [1] 13.5 17.0 20.5 24.0 27.5 31.0 34.5 38.0 41.5 45.0 48.5
```

```
w[!w<10]
```

```
 [1] 10.0 13.5 17.0 20.5 24.0 27.5 31.0 34.5 38.0 41.5 45.0
```

```
w[w%%2==0]
```

```
[1] 10 24 38
```

```
w[w%%2==1]
```

```
[1]  3 17 31 45
```

```
w>30
```

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

# Subvectors

```
w[w%%2==1]
```

```
[1]  3 17 31 45
```

```
w>30
```

```
 [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE
[12]  TRUE  TRUE  TRUE
```

# Subvectors

If we evaluate one vector in another vector, this will evaluate the position that meet the condition, also just can be possible if the two vectors has the same length

```
x = c(1,7,4,2,4,8,9,2,0)
y = c(5,2,-3,-7,-1,4,-2,7,1)
x
y>0
x[y>0]
```

```
[1] 1 7 4 2 4 8 9 2 0
[1]  TRUE  TRUE FALSE FALSE FALSE  TRUE FALSE  TRUE  TRUE
[1] 1 7 8 2 0
```

# Conditional

- `which(x meets condition)`: to obtain the indexes of the $x$ vector entries that satisfy the given condition
- `which.min(x)`: gives us the first position in which the vector $x$ takes its minimum value
- ‘`which(x==min(x))` ”: gives all the positions in which the vector $x$ takes its minimum values
- `which.max(x)`: gives us the first position in which the vector $x$ takes its maximum value
- `which(x ==max(x))`: gives all the positions in which the vector $x$ takes its maximum values