

# 1. R Foundations - Atomic Vectors

CT5102 - J. Duggan

# R

- R's mission is to enable the best and most thorough exploration of data possible (Chambers 2008).
- It is a dialect of the S language, developed at Bell Laboratories
- ACM noted that S “will forever alter the way people analyze, visualize, and manipulate data”
- See <https://github.com/JimDuggan/CT5102>

```
v <- 1:10
```

```
v
```

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

```
summary(v)
```

```
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##      1.00   3.25   5.50   5.50   7.75   10.00
```

# R Studio IDE (also available through <https://rstudio.cloud>)

## R Code

```
01 Introduction.R x
Source on Save  Run  Source
83 s4<-sample(3,20,replace=TRUE)
84 s2=s4
85
86 # Gather data on the frequency of (whole) numbers in a vector
87 t1<-table(s1)
88 props<-prop.table(t1)
89
90
91
92
93
94
95
96
97
98
99
100:1 (Top Level) s R Script
```

## Environment/State

CT5102		
Environment History Git		
Global Environment		
Values		
b1	logi [1:2]	FALSE TRUE
b2	logi [1:5]	FALSE TRUE FALSE TRUE FALSE
c1	chr [1:5]	"Odd" "Even" "Odd" "Even" "Odd"
ind	2L	
index	5L	
props	table [1:3(1d)]	0.4 0.3 0.3
r	24	
s	num [1:101]	51.4 55 57.2 57.3 58.6 ...
s1	int [1:20]	2 1 1 3 1 3 2 1 1 1 ...
s2	int [1:20]	2 3 2 2 2 2 1 2 2 1 ...

```
Console ~/Desktop/GitHub/CT5102/
> s2=s4
[1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
> t1<-table(s1)
> props<-prop.table(t1)
> props
s1
 1  2  3
0.4 0.3 0.3
>
>
```

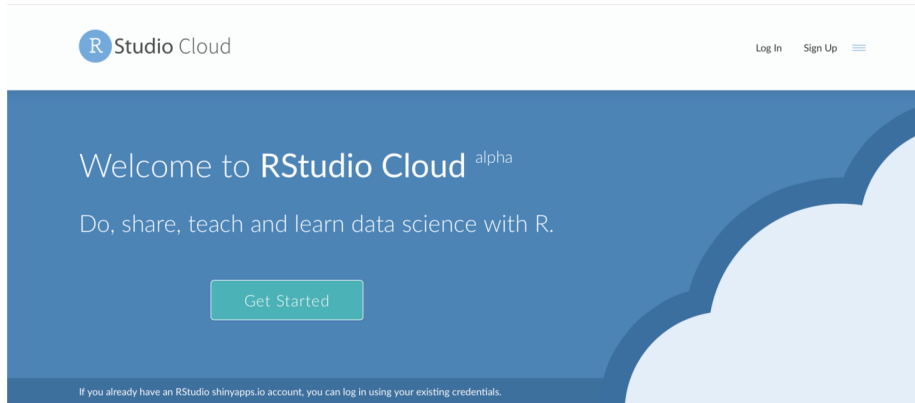
Files Plots Packages Help Viewer			
New Folder Delete Rename More			
Home Desktop GitHub CT5102 01 Vectors			
Name		Size	Modified
..			
01 Introduction.R		1.9 KB	Sep 3, 2015, 2:23 PM
01 Vectors.pdf		492 KB	Sep 3, 2015, 2:26 PM

## Interactive console

## File System

Figure 1: R Studio IDE  
1. R Foundations - Atomic Vectors

# Setup an Account on rstudio.cloud



# Create a project

The screenshot displays the RStudio Cloud web interface. On the left is a sidebar with navigation links: 'Spaces' (containing 'Your Workspace' and '+ New Space'), 'Learn' (containing 'Guide', 'What's New', 'Primers', 'Cheat Sheets', and 'Feedback and Questions'), and 'Info' (containing 'Terms and Conditions'). The main content area is titled 'Your Workspace' and has tabs for 'Projects' (selected) and 'Info'. Below the 'Projects' tab, the heading 'Your Projects' is followed by a large grey box containing the text 'No Projects'. A 'New Project' button is located above this box, and its dropdown menu is open, showing two options: '+ New Project' and 'New Project from Git Repo'. On the right side of the interface, there is a user profile for 'Jim Duggan' and a section titled 'Options' with a close button. This section includes a 'Search Projects' search bar and a 'Sort Projects' section with two radio buttons: 'By name' (selected) and 'By date created'. Below this is a 'Capacity' section with a description: 'This your personal workspace, where you can create a virtually unlimited number of projects.' and a link to 'Learn more about Your Workspace in the Guide.'

# RStudio ready for use

The screenshot displays the RStudio Cloud interface for a workspace named "CT1100 Workspace / Project 101". The user is identified as "Jim Duggan".

**Left Sidebar (Spaces):**

- Your Workspace
- CT1100 Workspace (selected)
- New Space

**Learn Section:**

- Guide
- What's New
- Primers
- Cheat Sheets
- Feedback and Questions

**Info Section:**

- Terms and Conditions
- System Status

**Main Console:**

```
R version 3.6.0 (2019-04-26) -- "Planting of a Tree"
Copyright (C) 2019 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'licence()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> |
```

**Environment Pane:**

Global Environment -

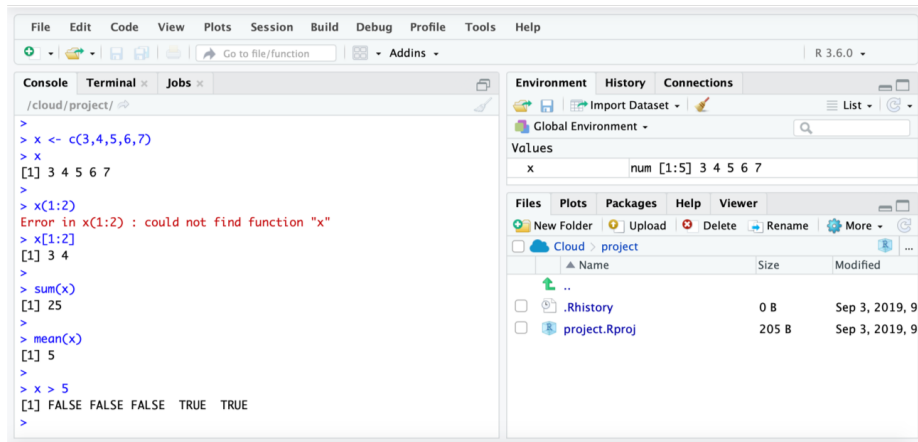
Environment is empty

**Files Pane:**

Name	Size	Modified
..		
.Rhistory	0 B	Sep 3, 2019, 9:20 AM
project.Rproj	205 B	Sep 3, 2019, 9:20 AM

# Run code in console.

- R allows you process the data with function calls



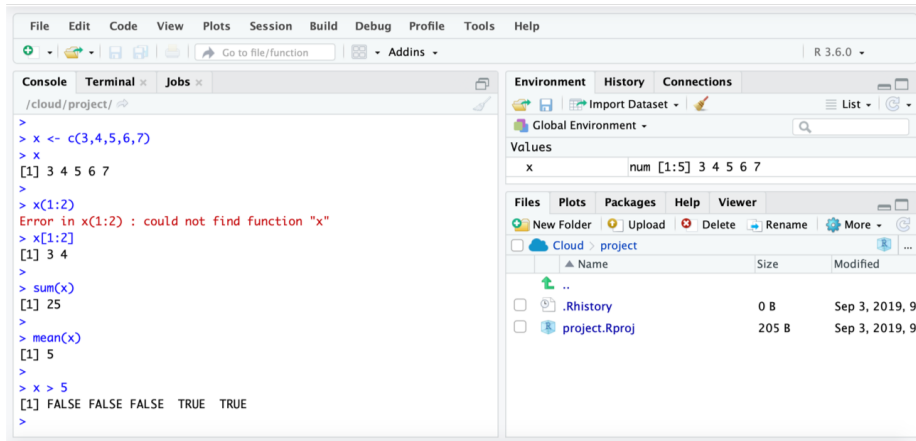
The screenshot displays the RStudio interface. The top menu bar includes File, Edit, Code, View, Plots, Session, Build, Debug, Profile, Tools, and Help. Below the menu is a toolbar with icons for saving, opening, and navigating files, along with a search bar and a dropdown for 'Addins'. The main window is divided into two panes. The left pane, titled 'Console', shows the following R code and its output:

```
>  
> x <- c(3,4,5,6,7)  
> x  
[1] 3 4 5 6 7  
>  
> x(1:2)  
Error in x(1:2) : could not find function "x"  
> x[1:2]  
[1] 3 4  
>  
> sum(x)  
[1] 25  
>  
> mean(x)  
[1] 5  
>  
> x > 5  
[1] FALSE FALSE FALSE TRUE TRUE  
>
```

The right pane is titled 'Environment' and shows the 'Global Environment'. It displays the variable 'x' with its values: num [1:5] 3 4 5 6 7. Below this, there is a 'Files' pane showing the project structure. It includes a 'New Folder' button, an 'Upload' button, and a 'Delete' button. The file list shows the following files:

	Name	Size	Modified
<input type="checkbox"/>	..		
<input type="checkbox"/>	.Rhistory	0 B	Sep 3, 2019, 9
<input type="checkbox"/>	project.Rproj	205 B	Sep 3, 2019, 9

# Challenge 1.1 - Replicate the following in RStudio Cloud



The screenshot displays the RStudio Cloud interface. The top menu bar includes File, Edit, Code, View, Plots, Session, Build, Debug, Profile, Tools, and Help. The toolbar contains icons for creating a new file, opening a file, saving, and navigating to a file/function. The version is R 3.6.0.

The left pane shows the Console with the following session:

```
>
> x <- c(3,4,5,6,7)
> x
[1] 3 4 5 6 7
>
> x[1:2]
Error in x[1:2] : could not find function "x"
> x[1:2]
[1] 3 4
>
> sum(x)
[1] 25
>
> mean(x)
[1] 5
>
> x > 5
[1] FALSE FALSE FALSE TRUE TRUE
>
```

The right pane shows the Environment pane with the Global Environment. The Values table is as follows:

Variable	Value
x	num [1:5] 3 4 5 6 7

The Files pane shows the project structure:

Name	Size	Modified
..		
.Rhistory	0 B	Sep 3, 2019, 9
project.Rproj	205 B	Sep 3, 2019, 9



# R Data Types

	Homogenous	Heterogenous
1d	<b>Atomic Vector</b>	List
2d	Matrix	Data Frame/Tibble
nd	Array	

- The basic data structure in R is the Vector
- Vectors come in two flavours:
  - Atomic vectors
  - Lists
- With atomic vectors, all elements have the same type: logical, integer, double (numeric) or character
- **typeof()** **str()** functions useful

# Atomic Vectors - Examples

```
dbl_var <- c(2.9, 3.1, 4.8)
typeof(dbl_var)
```

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

```
int_var <- c(0L, 1L, 2L)
typeof(int_var)
```

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

```
log_var <- c(TRUE, TRUE, FALSE, T, F)
typeof(log_var)
```

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

```
str_var <- c("Dublin", "London", "Edinburgh")
typeof(str_var)
```

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

# str() function useful

```
str(dbl_var)
```

```
##  num [1:3] 2.9 3.1 4.8
```

```
str(int_var)
```

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

```
str(log_var)
```

```
##  logi [1:5] TRUE TRUE FALSE TRUE FALSE
```

```
str(str_var)
```

```
##  chr [1:3] "Dublin" "London" "Edinburgh"
```

## Creating Sequences : and seq() function

```
v1 <- 1:10
```

```
v1
```

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

```
v2 <- 10:20
```

```
v2
```

```
## [1] 10 11 12 13 14 15 16 17 18 19 20
```

```
v3 <- seq(20, 30, by=1)
```

```
v3
```

```
## [1] 20 21 22 23 24 25 26 27 28 29 30
```

# Creating Vectors of fixed size (in advance)

```
v1 <- vector(mode="numeric", length=20)
```

```
v1
```

```
## [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
v2 <- vector(mode="logical", length=5)
```

```
v2
```

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

# Coercion of atomic vectors

- All elements of an atomic vector **MUST** be of the same type
- When different type are combined, they will be coerced into the most flexible types

	<b>logical</b>	<b>integer</b>	<b>numeric</b>	<b>character</b>
<b>logical</b>	logical	integer	numeric	character
<b>integer</b>	integer	integer	numeric	character
<b>numeric</b>	numeric	numeric	numeric	character
<b>character</b>	character	character	character	character

# Coercion Examples

```
v1 <- c(10, 20, TRUE)
v1
```

```
## [1] 10 20 1
```

```
typeof(v1)
```

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

```
v2 <- c(10, 20, "True")
v2
```

```
## [1] "10" "20" "True"
```

```
typeof(v2)
```

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

## Challenge 1.2

Determine the types for each of the following vectors

```
v1 <- c(1L, T, FALSE)
v2 <- c(1L, T, FALSE, 2)
v3 <- c(T, FALSE, 2, "FALSE")
v4 <- c(2L, "FALSE")
v5 <- c(0L, 1L, 2.11)
```



# Subsetting Atomic Vectors

- Subsetting data is a key activity in data science
- R's subsetting operators are powerful and fast
- For atomic vectors, the operator `[]` is used
- In R, the index for a vector starts at 1

```
x <- c( 2.1, 4.2, 3.3, 5.4)
```

```
x
```

```
## [1] 2.1 4.2 3.3 5.4
```

```
x[1]
```

```
## [1] 2.1
```

```
x[c(1,4)]
```

```
## [1] 2.1 5.4
```

# Subsetting Vectors - (1) Positive Integer

Positive integers return elements at the specified position

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

```
x
```

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

```
x[5]
```

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

```
x[8:10]
```

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

## Subsetting Vectors - (2) Negative Integer

Negative integers omit elements at specified positions

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

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

```
x[-5]
```

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

```
x[-(8:10)]
```

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

```
x[-(2:10)]
```

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

## Subsetting Vectors - (3) Logical Vectors

- Select elements where the corresponding logical value is TRUE.
- This approach supports recycling

```
x <- 1:5
```

```
x
```

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

```
x[c(F,T,T,T,T)]
```

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

```
x[c(F,T)]
```

```
## [1] 2 4
```

# Logical Vectors - Can be formed with logical expressions

```
x <- 1:5
```

```
x
```

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

```
lx <- x < 2
```

```
lx
```

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

```
x[lx]
```

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

```
x[x>2]
```

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

# Subsetting Vectors - (4) Using character vectors

Return elements with matching names

```
x <- 1:5  
names(x) <- c("a", "b", "c", "d", "e")  
x
```

```
## a b c d e  
## 1 2 3 4 5
```

```
x["a"]
```

```
## a  
## 1
```

```
x[c("a", "e")]
```

```
## a e  
## 1 5
```

## Challenge 1.3

- Create an R vector of squares of 1 to 10
- Find the minimum
- Find the maximum
- Find the average
- Subset all those values greater than the average

# Vectorisation

- A powerful feature of R is that it supports vectorisation
- Functions can operate on every element of a vector, and return the results of each individual operation in a new vector.

```
x <- c(1,4,9,16,25)
```

```
x
```

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

```
y <- sqrt(x)
```

```
y
```

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



# Vectorisation

**Input Vector**



`sqrt()`

**Output Vector**



**Figure 2:** Vectorisation in R

# Vectorised if/else

Vectors can also be processed using the vectorized `ifelse(b,u,v)` function, which accepts a boolean vector `b` and allocates the element-wise results to be either `u` or `v`.

```
v1 <- 1:5  
  
ans <- ifelse(v1 %% 2 == 0, "Even", "Odd")  
ans  
  
## [1] "Odd" "Even" "Odd" "Even" "Odd"
```

# Sample Function

`sample` takes a sample of the specified size from the elements of `x` using either with or without replacement.

## Usage

```
sample(x, size, replace = FALSE, prob = NULL)
```

```
sample.int(n, size = n, replace = FALSE, prob = NULL)
```

## Arguments

- `x` Either a vector of one or more elements from which to choose, or a positive integer. See 'Details.'
- `n` a positive number, the number of items to choose from. See 'Details.'
- `size` a non-negative integer giving the number of items to choose.
- `replace` Should sampling be with replacement?
- `prob` A vector of probability weights for obtaining the elements of the vector being sampled.

**Figure 3:** Sample function in Base R

```
s <- sample(c("Y", "N"), 10, prob=c(.2, .8), repl=T)
```

# NA Symbol in R (Not available)

- In a project of any size, data is likely to be incomplete due to
  - Missed survey questions
  - Faulty equipment
  - Improperly coded data
- In R, missing data is represented by the symbol NA

```
x <- 1:5  
x[3] <- NA  
x
```

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

```
sum(x)
```

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

```
sum(x, na.rm=TRUE)
```

```
## [1] 12
```

# Testing for NA? Need is.na() function

- The function `is.na()` indicates which elements are missing
- Returns a logical vector, the same size as the input vector

```
x
```

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

```
is.na(x)
```

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

```
which(is.na(x)) # get the location of NA
```

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

```
x[!is.na(x)] # Exclude all NAs from result
```

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

## Challenge 1.4

- Create a vector of 100 numbers
- Set 10 of these to NA (random)
- Print the locations of the missing values
- Hint: Check out the R function **which()**

# Programming in R

- R is a block-structured language, where blocks are delineated by `{}`
- Statements separated by newline characters, or with semicolon
- Variable types are not declared (similar to JavaScript)

# Loops in R

Method	Syntax	Comments
loop over the elements	for (x in xs)	not a good choice for a for loop because it leads to inefficient ways of saving output
loop over the numeric indices	for (i in seq_along(xs))	Allows you to to create the space you'll need for the output and then fill it in.
loop over the names	for (nm in names(xs))	

```
x<-1:4
y<-vector(mode="numeric",length = length(x))
for(i in seq_along(x)){
  y[i] <- x[i]
}
y
```

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



# If statement in R

```
x<-4
y <- "Unknown"
if(x %% 2 == 0){
  y <- "Even Number"
}
y
```

```
## [1] "Even Number"
```

# If-else statement in R

```
x<-5
y <- "Unknown"
if(x %% 2 == 0){
  y <- "Even Number"
}else{
  y <- "Odd Number"
}
y
```

```
## [1] "Odd Number"
```

- It is important to note that **else** must be in the same line as the closing braces of the if statements.

# If-else-if

```
x<- -10

if(x > 0){
  print("Positive Number...")
}else if (x < 0){
  print("Negative Number...")
} else{
  print("Number is zero...")
}
```

```
## [1] "Negative Number..."
```

# sprintf() function for printing to console

A wrapper for the C function **sprintf**, that returns a character vector containing a formatted combination of text and variable values.

```
sprintf("%s is %f feet tall", "Sven", 7.1)
```

```
## [1] "Sven is 7.100000 feet tall"
```

```
sprintf("%f", pi)
```

```
## [1] "3.141593"
```

```
sprintf("%.3f", pi)
```

```
## [1] "3.142"
```

```
sprintf("%1.0f", pi)
```

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

# Summary

- Atomic vectors, key type in R
- All elements are the same type (coercion)
- Different ways to filter, including logical vectors
- `is.na()` to check for symbol NA (not available)
- Vectors support vectorised operations. e.g., **`sqrt(1:10)`**