R Objects/data structures

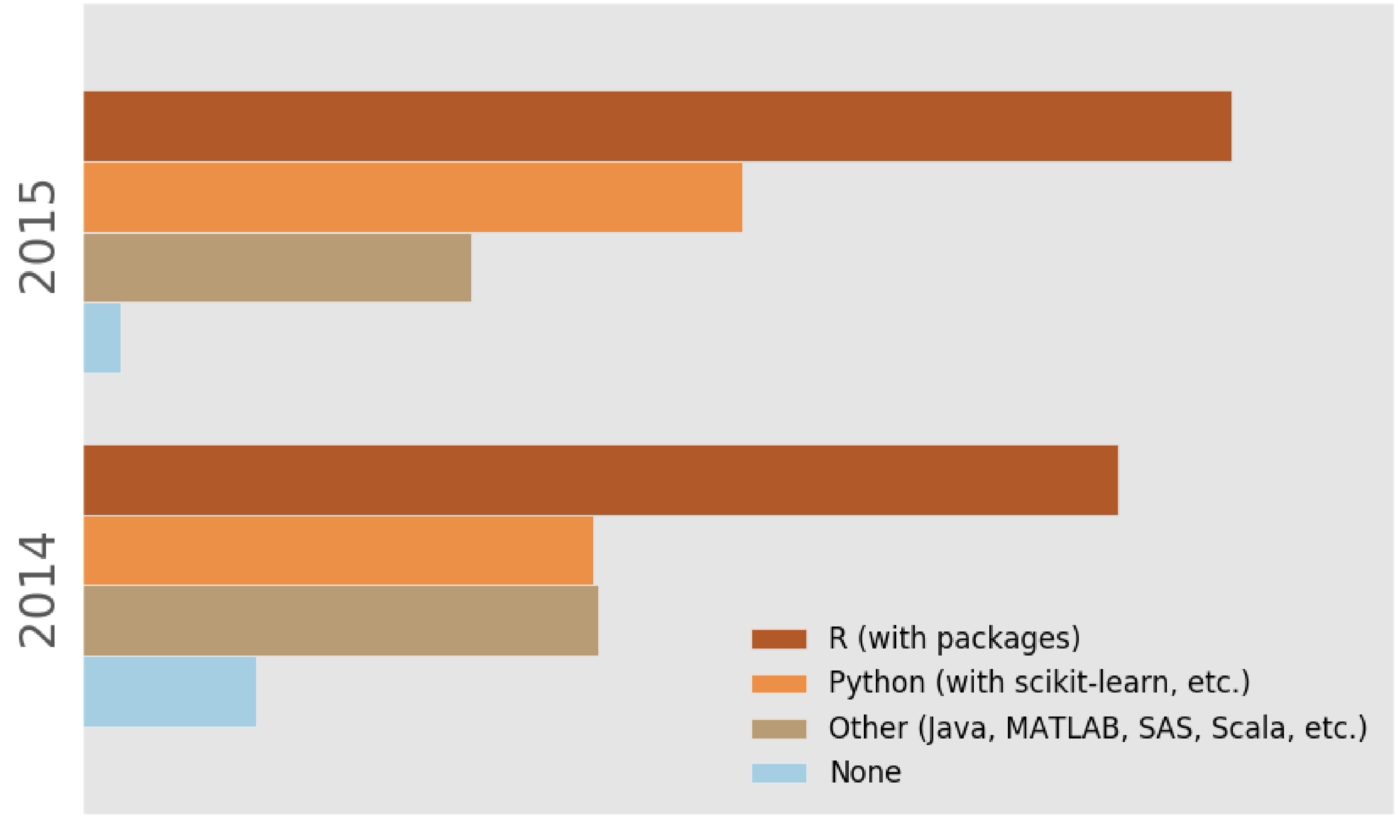
Benjamin Soibam, PhD

CS 5301 – Programming Foundations for Data Analytics

Reference for the slides

- Mainly from Part II of *Hands on Programming with R* by Garrett Grolemund
- Section 20 of R for Data Science Book

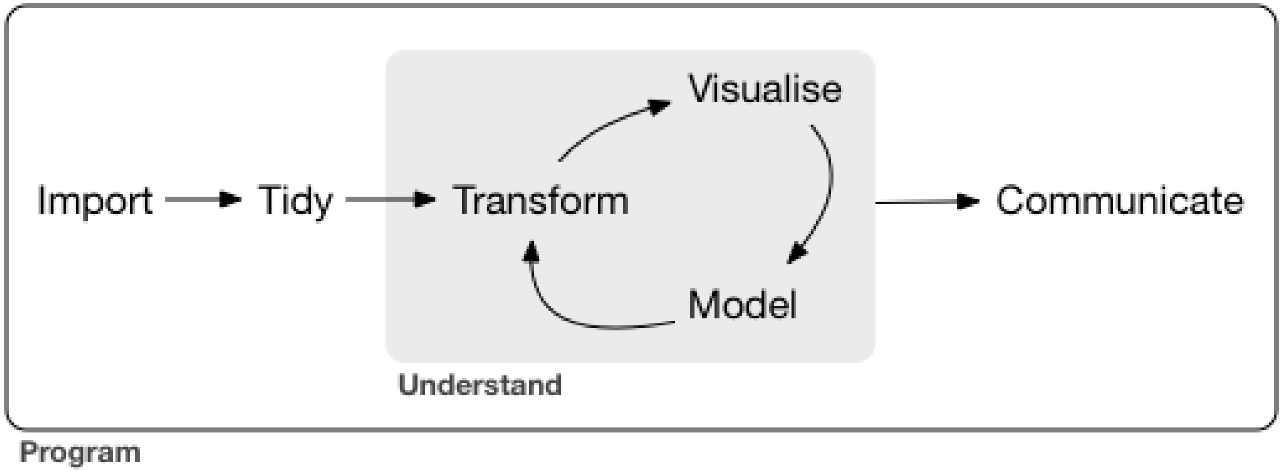
Two main programming essentials for Data Analytics: **R and Python**



http://blog.udacity.com/2016/04/languages-and-libraries-for-machine-learning.html

R for Data Science

- Data Science is a huge field.



What is this course about

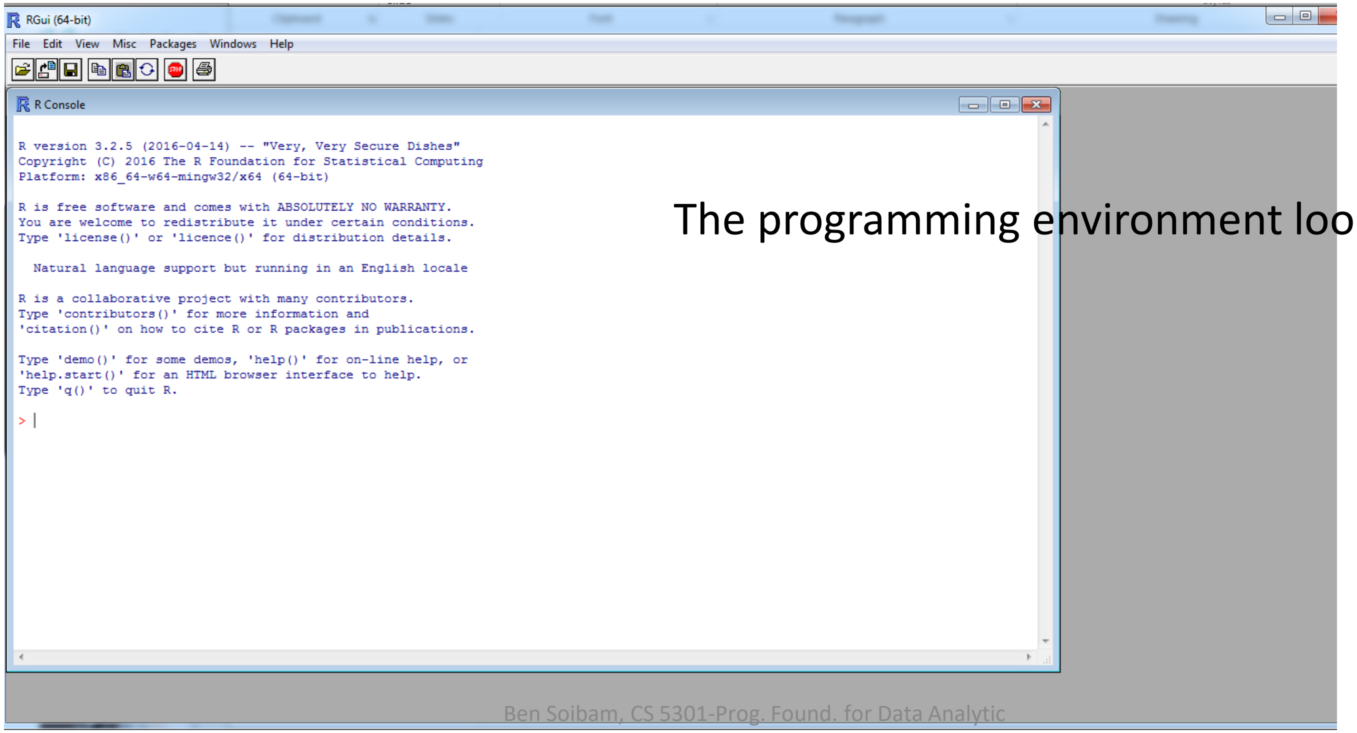
- The purpose of this course is to provide programming essentials or foundations required for more advanced data analytics courses
- Most data analytics courses use R (mostly) or Python

What this course is not about

- You will not learn data mining or analytics methods or algorithms
- You will not learn specific techniques related to BIG DATA

Download and install R

- To download R, go to CRAN, the **c**omprehensive **R** **a**rchive **n**etwork
- https://cloud.r-project.org
- Better to get the latest version



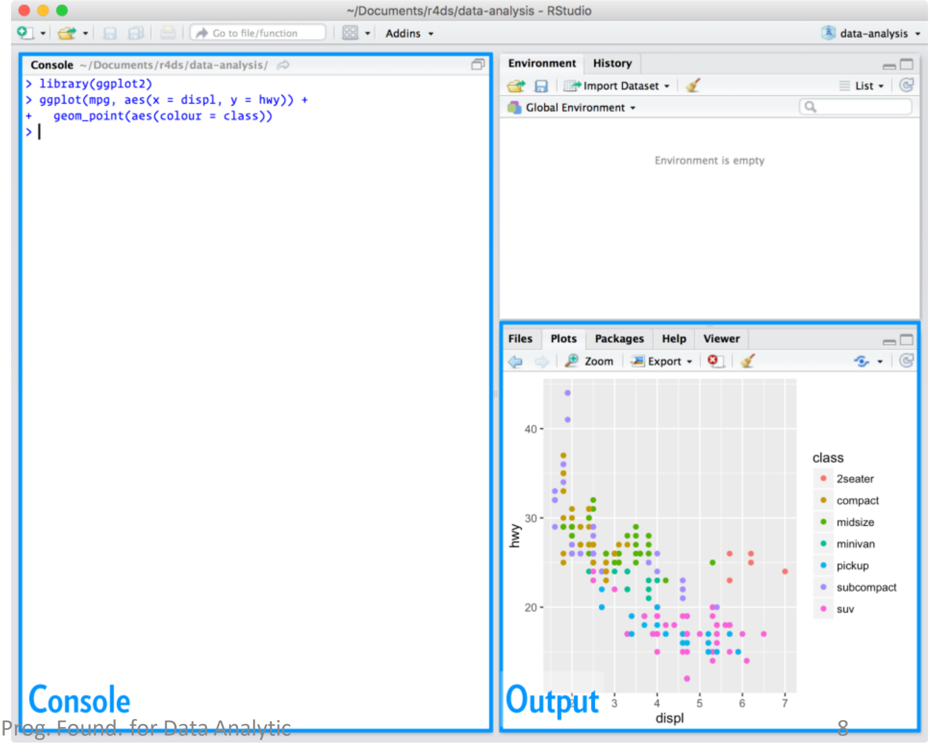
```
R version 3.2.5 (2016-04-14) -- "Very, Very Secure Dishes"
Copyright (C) 2016 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

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

  Natural language support but running in an English locale

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.

> |
```

The programming environment looks like this

Download and install RStudio

- RStudio is an integrated development environment, or IDE, for R programming.
- Download and install it from http://www.rstudio.com/download.

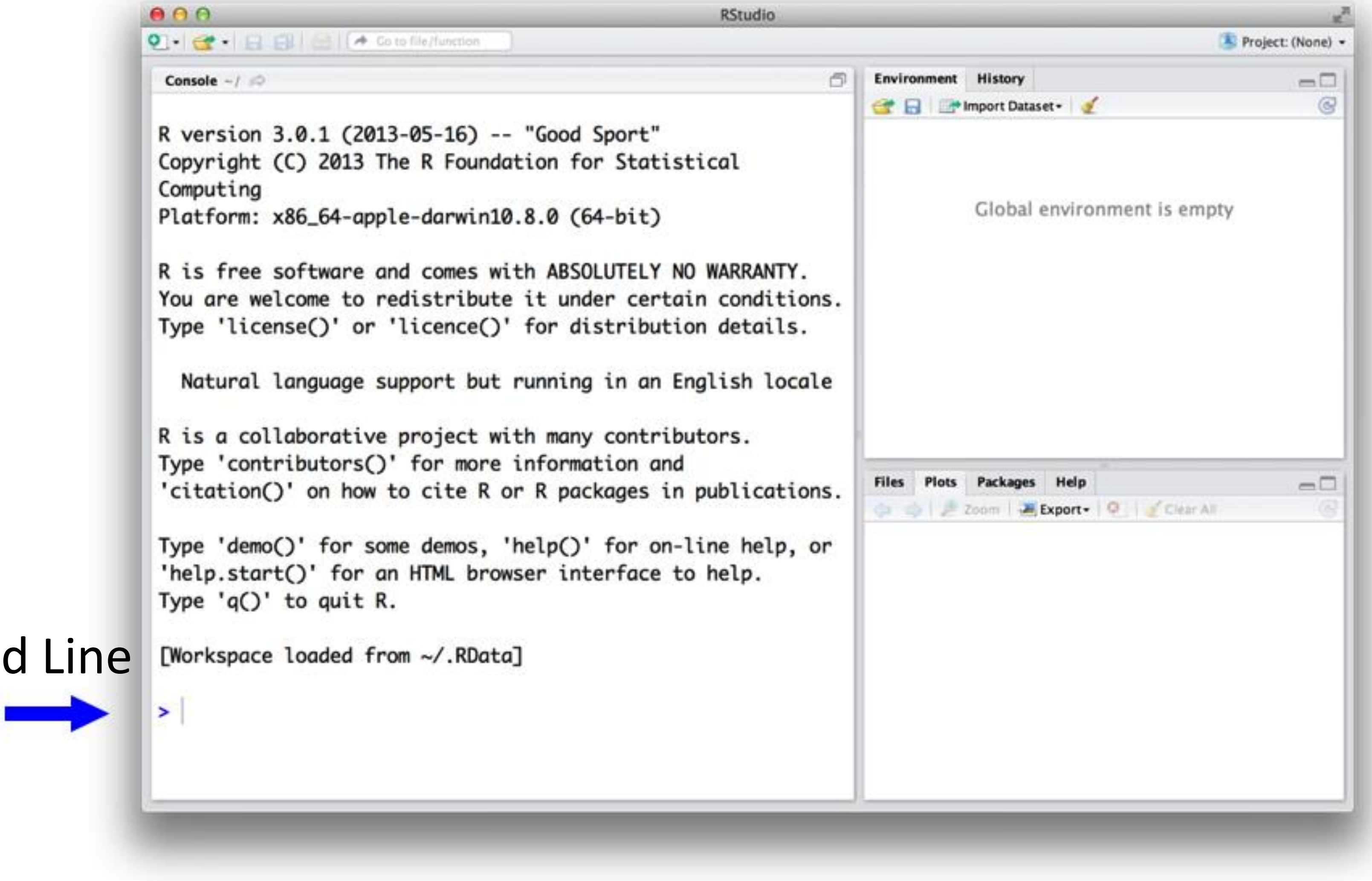
When you start RStudio, you'll see two key regions in the interface:



```
> library(ggplot2)
> ggplot(mpg, aes(x = displ, y = hwy)) +
+   geom_point(aes(colour = class))
> |
```

Environment is empty

Command Line



```
R version 3.0.1 (2013-05-16) -- "Good Sport"
Copyright (C) 2013 The R Foundation for Statistical
Computing
Platform: x86_64-apple-darwin10.8.0 (64-bit)

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

  Natural language support but running in an English locale

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.

[Workspace loaded from ~/.RData]

> 
```

Simple Arithmetic operations

```
> 1 + 1
[1] 2
```

[1] means this line begins with the first value in your result

```
> 100:120
 [1] 100 101 102 103 104 105 106 107 108 109 110   111
112 113 114 115 116 117 118 119 120
```

The colon “:” means generate a sequence of numbers (or a vector which a one-dimensional set of numbers)

```
> 10:25
 [1] 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
```

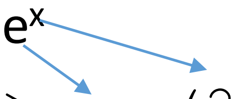
Simple Arithmetic operations

```
>10 + 2
[1] 12
>12 * 3
[1] 36
>36 - 6
[1] 30
>30 / 3
[1] 10
> 23/4
[1] 5.75
```

Simple Arithmetic operations

```
> 2**3  (power)
[1] 8
> 3.2**2
[1] 10.24
```

Exponential function



e^x

```
> exp(2)
[1] 7.389056
```

Simple Arithmetic operations

logarithm function

Log$_x^y$ x is base, y is exponent

```
> log(3,2)    # 3 is base, 2 is the exponent
[1] 1.584963
```

For bases 2 and 10, you can also do

```
> log2(3)
[1] 1.584963
> log10(3)
[1] 0.4771213
```

Arithmetic expressions

Arithmetic operation within parenthesis will be done first

```
>1 + (3+4)/2
[1] 4.5

> ((4+5)/2)**(2+3) # ((4+5)/2)^(2+3)
[1] 1845.281
```

The ‘Esc’ key

- If ever get stuck or something is taking too long to run. You can simply hit the ‘Esc’ key and return to the command prompt/line

getwd() : get working directory

- Returns an absolute filepath representing the current working directory of the R process;

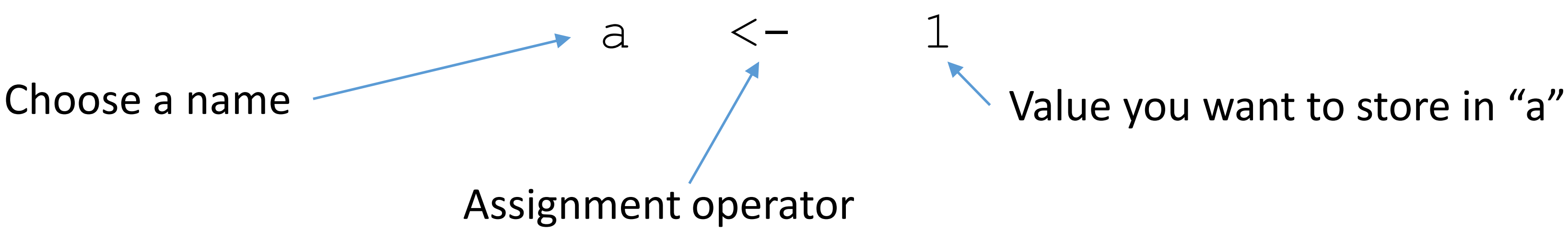
setwd(dir) : get working directory

- is used to set the working directory to dir

- Example:
- setwd("C:/Users/soibamb/Dropbox/Teaching/CS5301/")
- This is for windows

Objects

- It’s a name or a variable in R that allows you to save data so that you can use it later.
- The line below will store the value 1 in the R variable/object “a”.



```
a    <-     1
```

Choose a name

Assignment operator

Value you want to store in “a”

```
> a <- 1
> a
[1] 1
> a + 3
[1] 4
>print(a)
[1] 1
```

R Objects

- Used to store information while running a program
  - Atomic Vectors
    - Stores a collection of data of same type.
    - For example a collection of numbers, a collection of words.
    - Doesn’t include relational information
  - List
    - Stores a collection of data of same or different type.
    - For example a collection of numbers, and words.
    - Doesn’t include relational information
  - Matrices
    - Stores data in tabular form. Organizes a vector in a tabular form (rows and columns)
  - **Data Frames (or tibble)**
    - **Stores data in a tabular form (rows and columns). Relational data**
    - **Can assign names/annotations to rows and columns**

R Objects

- Used to store information outside of a program
  - Files
    - Stores ONE R object
  - Rdata format
    - Stores multiple R objects

Atomic vector

- An atomic vector is a simple vector of data (multiple items)
- Creating atomic vector

Think the **‘c’** in terms of “combining”

```
x <- c(1,1,3,0,-5,10)
```

x stores 6 different items in memory

	1	**1**	**3**	**0**	**-5**	**10**
Index	1	2	3	4	5	6

- use `is.vector(x)` to test if the object `x` is a vector

```
>is.vector(x)
   TRUE
```

Atomic vector

- You can also create vector with one item

```
> x <- c(1)
```

- You can also create an empty vector

```
> y <- c()
```

- To find how many items are there in a vector, use `length`

```
 > x <- c(1,2,2)
> length(x)
 3
```

Atomic vectors

- Depending on the items stored in a vector, R supports different types of atomic vector

- double (numeric)
- Integer
- Characters
- Logicals

Double vector

> A double vector stores regular numbers. The numbers can be +ve or –ve.

```
die <- c(1, 2, 3, 4, 5, 6)
die

typeof(die)  # gives you what data type
# “double”
```

Integer vector

- You can create an integer in R by typing a number followed by “L”

```
> num <- c(-1L, 2L, 4L)
```

```
>typeof(num)
[1] "integer“
```

- Main difference between the two statements below is the amount of memory used to store them.

```
> num1 <- c(-1, 2, 4) # double (requires more memory)
> num2 <- c(-1L, 2L, 4L) #integer
```

Character vector

- A character vector that stores small pieces of text

```
> text <- c("Hello","World")
> text
[1] "Hello" "World“
>typeof(text)
[1] "character"
```

Logical vectors

- Logical vectors stores TRUEs and FALSEs. (Boolean data)

```
> 3 > 4
[1] FALSE
> logic <- c(TRUE, FALSE)
> logic
[1]  TRUE FALSE

> typeof(logic)
[1] "logical"
```

Other ways of vectors: seq

- Generate a sequence of numbers (with regular intervals) using R function **seq**

Starting number

maximal end number

Increment in the sequence

```
> x1 <-  seq(from=1, to = 10, by = 3)
> x2 <-  seq(from=1, to = 10, by = 2)
> x1
[1]  1  4  7 10
> x2
[1] 1 3 5 7 9
> seq(from=1.2,to=10,by = 1.2)
[1] 1.2 2.4 3.6 4.8 6.0 7.2 8.4 9.6
```

Replicate Elements of Vectors: rep

- Replicate Elements of Vector

```
> x1 <- seq(from = 1, to = 10, by = 4)
> x1
[1] 1 5 9
> y1 <- rep(x1,3)
> y1
[1] 1 5 9 1 5 9 1 5 9
```

To be replicated

# of times to be replicated

- Creating a vector containing 10 zeros

```
> x2 <- 0
> y2 <- rep(x2,10)
> y2
 [1] 0 0 0 0 0 0 0 0 0 0
```

Accessing (Sub setting) vector elements

```
> x1 <- seq(from = 1, to = 10, by = 1)
> x1
 [1]  1  2  3  4  5  6  7  8  9 10
```

- Extracting item located in index = 3

```
> y1 <- x1[3]
> y1
[1] 3
```

- Extracting items located in indices 2 to 5

```
> y1 <- x1[2:5]
> y2 <- x1[2:5]
> y2
[1] 2 3 4 5
```

Extracting items located in indices 1, 4, and 5

```
> y3 <- x1[c(1,4,6)]
> y3
[1] 1 4 6
```

Accessing (Sub setting) vector elements

```
> x1 <- seq(from = 1, to = 10, by = 1)
> x1
 [1]  1  2  3  4  5  6  7  8  9 10
```

- removing item located in index = 3

```
> y1 <- x1[-3]
> y1
[1] 1 2 4 5 6 7 8 9 10
```

- Removing items located in indices 2 to 5

```
> y1 <- x1[-c(2:5)]
> y1
 [1] 1  6  7  8  9 10
```

removing items located in indices 1, 4, and 5

```
> y3 <- x1[-c(1,4,6)]
> y3
[1] 2 3 5 7 8 9 10
```

Modifying vectors

```
> vec <- c(0, 0, 0, 0, 0, 0)

# change the first item in the vector to 1000
> vec[1] <- 1000
#change multiple items in a vector
> vec[c(1,3,5)] <- c(1,1,1)
#add 1 to the selected items
>vec[4:6] <- vec[4:6] + 1
# create values that don’t exist
>vec[7] <- 0
# change the multiple items in the vector to 2
> vec[c(1,3,4)] <- 2
```

Operations between Vectors

- Adding/multiplying/dividing two vectors of same length
- This will perform element wise operation

```
> c(1,2,3) * c(2,3,4)
[1]  2  6 12
> c(1,2,3) / c(2,3,4)
[1] 0.5000000 0.6666667 0.7500000
> c(1,2,3) + c(2,3,4)
[1] 3 5 7
```

Operations between Vectors

- Adding/multiplying/dividing two vectors of **different length**
- The length of the longer vector should be a multiple of the length of the shorter vector
- The shorter vector will be self concatenated to match the length of the longer vector
- If the length of the longer vector is not a multiple of the length of the shorter vector, there will be an error

`> c(1,2,3,4) * c(2)` is same as `c(1,2,3,4) * c(2,2,2,2)`

`> c(1,2,3,4) * c(2,1)` is same as `c(1,2,3,4) * c(2,1,2,1)`

`> c(1,2,3,4) * c(2,1,2)` will give an error.

Combining or concatenating vectors

```
> x <- c(1,2,3,4)
> y <- c(100,2)
> z <- c(x,y)
> z
[1]   1   2   3   4 100   2
```

Attributes

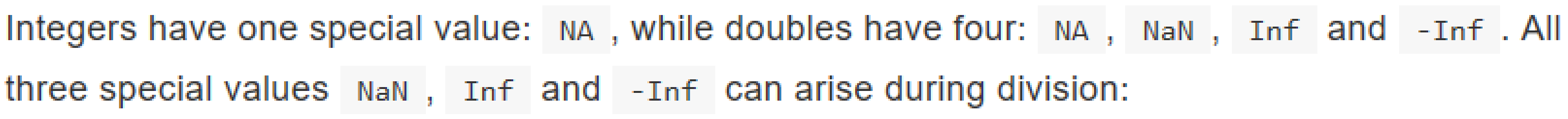
- An attribute is a piece of information that you can attach to an atomic vector (or any Robject).
- The attribute won’t affect any of the values in the object. You can think of an attribute as “metadata”

```
die<- c(1,2,3,4,5,6)

attributes(die)

## NULL
```

Attaching attribute

```
> names(die) <-
c("one","two","three","four","five","six")

> die
  one   two three  four  five   six
    1     2     3     4     5     6
```

Special Values in a vector



Integers have one special value: `NA`, while doubles have four: `NA`, `NaN`, `Inf` and `-Inf`. All three special values `NaN`, `Inf` and `-Inf` can arise during division:

```
c(-1, 0, 1) / 0
#> [1] -Inf NaN Inf
```

Dealing with these special values will be discussed in more detail in the future lectures.

Missing Value in a vector

- In some cases, an atomic vector may contain missing value. It is indicated by ‘NA’

```
x <- c(NA, 1, 2)
```

Dealing with missing values will be discussed in more detail in the future lectures.

Coercion

- Vector should contain items of the same data type.
- If you try to create something like the following, the lower ranking type will be coerced (converted) to the higher ranking type.

```
x <- c(1,2,”hello”)
```

The hierarchy of coercion

```
Logical < integer < numeric (double) < character
```

Coercion

The hierarchy of coercion

```
Logical < integer < numeric (double) < character
```

```
> x <- c(1,2,"hello")
> x
[1] "1"     "2"     "hello" # converted to character

> c(TRUE, 1.5, FALSE)
[1] 1.0 1.5 0.0  # TRUE is converted to 1 and FALSE to 0

> c(TRUE, "this_char")
[1] "TRUE"      "this_char" # logical TRUE is converted to character
”TRUE”
```

Explicit coercion

- User can explicitly convert the data type of a vector

- Character to numeric/integer

```
> as.integer(c("1", "2"))
[1] 1 2
> as.numeric(c("1", "2"))
[1] 1 2
> as.numeric(c("1", "er"))
[1]  1 NA
Warning message:
NAs introduced by coercion
```

Explicit coercion

- User can explicitly convert the data type of a vector

- numeric/integer to character

```
> as.character(c(1,3,4))
[1] "1" "3" "4"
```