STAT 3355
Introduction to Data Analysis

Lecture 03: R Basics II

Created by Dr. Qiwei Li
Presented by Dr. Octavious Smiley

Department of Mathematical Sciences
The University of Texas at Dallas

Last Class

- `R` and RStudio
  - We ask `R` a question, and `R` responds with an answer
- Mathematical operations
  - Arithmetic operators
  - Function operators
- Functions
  - Name, arguments, body, and output
- Logical operations
  - Condition statements

Learning Goals

- Know basic R data modes
  - Numeric
  - Integer
  - Character
  - Logical
- Know basic R data classes
  - A single variable
  - Data vector
    - Structured data vector
- Write a simple function in R with loop statement

Operators

- The process of creating a variable and giving it a value(s)
- Operators:
  - variable_name `<-`, `<<-`, or `=` value
  - value `->` or `->>` variable_name
- Tips:
  - Use one operator
  - My choice is

    variable_name `<-` value
  - Avoid use of `=` for its confusion with the logical operator `==`

Variable Names

variable_name <- value

- Rules:
  - "Must" start with a letter
  - Should not contain any mathematical and logical operators
  - Should not contain space
  - Case matters
- Tips:
  - Abbreviated but as long as clear
  - Use underline "_" or "." to separate two words
  - Capitalized the first letter in each word

Data Modes

variable_name <- value

- Numeric
- Integer: Ended with `L`
- Character/string: Quoted by `' '` or `" "`
- Logical: `TRUE` or `FALSE`
- ~~Complex~~

Data Classes

variable_name <- value

- Vector: A vector of multiple values with the same type `c()`
- Matrix: A matrix of multiple values with the same type `matrix()`
- Data Frame: A data frame of multiple values with the same type within each column `data.frame()`
- ~~List~~: A special type of vector, which can contain different data modes and/or data classes `list()`
- ~~Arrays~~

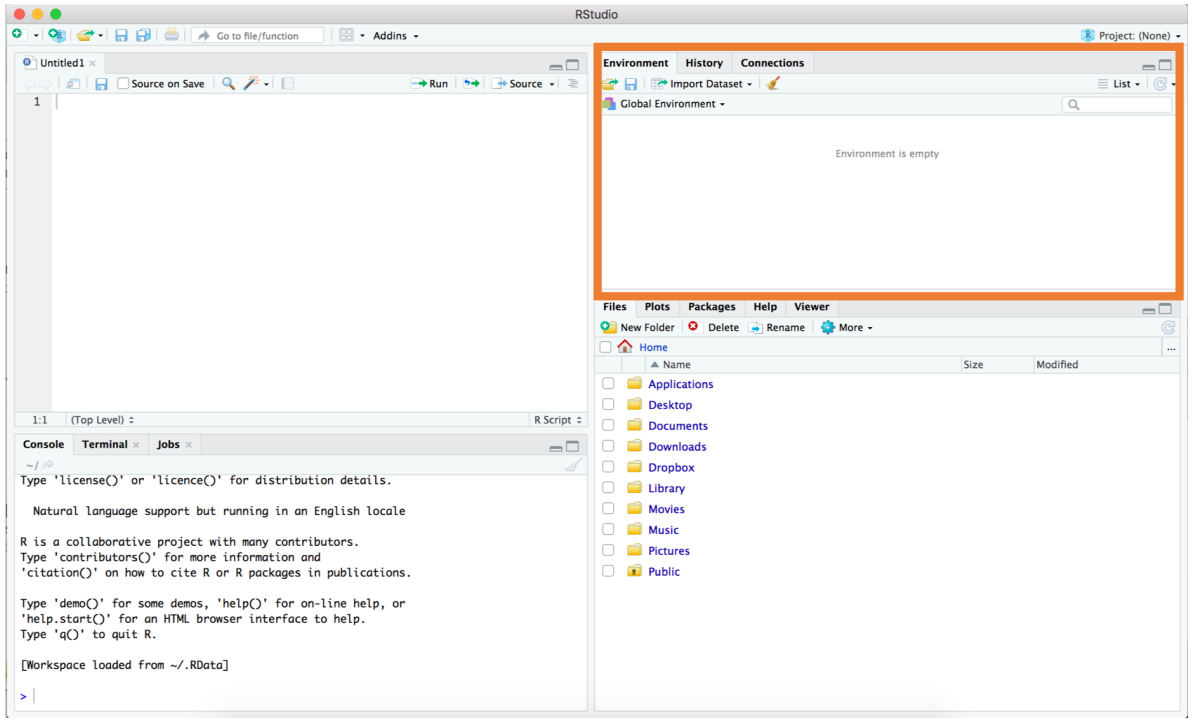
Single Variable Assignment

- Examples

```
# Numeric
x <- 5
x <- x + 5

# Integer
y <- 5L

# Character
z1 <- 'a'
z2 <- "hello.world"
z3 <- "5"

# Logical
b <- x == 5
```

View a Variable

- Typing in the variable name in the Console
- By using the function `print(variable_name)`
- Viewing in the Environment



Vector Assignment

- A list of variables that have the same type

$$\boldsymbol{x} = [x_1, \ldots, x_n]$$

- Input a vector via the function `c()`
  - x `<-` `c(`x_1`,...,`x_i`,...,`x_n`)`
  - Length is n, also called the number of entries
  - Each entry is a numeric, integer, character, or logical
  - If mixing the type, it will be coerced into one type with the order: character $>$ numeric $>$ integer $>$ logical
- Can contain only one entry: x `<-` `c(`x_1`)`

Vector Assignment

- Examples

```
x_num <- c(1.1, 2, 4, 6.3)
x_int <- c(1L, 2L, 4L, 6L)
x_chr <- c("a", "ac", "ab")
x_lgc <- c(TRUE, FALSE, TRUE)

# Automatic coercion
x <- c(1L, 2L, 1, 2, 1)
x <- c(1, 2, "1", 2, 1)
x <- c("1", "2L", 2L, 2, TRUE)
x <- c(x_num, x_int, x_chr, x_lgc)

# View the variable
x
print(x)
```

Vector Name

- Name a vector via the function `names(`x`)`
  - Automatic coercion to character
  - Always ensure the completeness of the data

Vector Name

- Examples

```
# Input the number of whales beachings per
    year in Texas during the 1990s
whales_tx <- c(74, 122, 235, 111, 292, 111,
    211, 133, 156, 79)

# Name the vector: automatic coercion
names(whales_tx) <- c(1990, 1991, 1992, 1993
    , 1994, 1995, 1996, 1997, 1998, 1999)
```

Single Entry Access

- Access a specified entry via
  - x`[`i`]`, where i is an integer between 1 and n
  - x`[-`i`]`, negative index returns all entries of the vector but the i-th one
  - x`[`a`]`, where a is an entry in `names(`x`)`
- Parentheses `()` mainly for functions and square brackets `[]` for data accessing

Single Entry Access

- Examples

```
# Input the number of whales beachings per
    year in Texas during the 1990s
whales_tx <- c(74, 122, 235, 111, 292, 111,
    211, 133, 156, 79)

# Name the vector: automatic coercion
names(whales_tx) <- c(1990, 1991, 1992, 1993
    , 1994, 1995, 1996, 1997, 1998, 1999)

# What is the number of whales in 1993?
whales_tx["1993"]
whales_tx[4]
```

Multiple Entries Access

- Access multiple entries via x`[`y`]`, where y is
  - A numeric data vector `c(`y_1`, ..., `y_m`)`, and each entry is an integer between 1 and n
  - A character data vector `c(`y_1`, ..., `y_m`)`, and each entry is an entry in `names(`x`)`
  - A logical data vector `c(`y_1`, ..., `y_n`)`, and each entry is a logical
    - Keep those entries with indices corresponding to `TRUE`
- x`[]` is equivalent to x

Multiple Entries Access

- Examples

```
# What is the number of whales in 1991 and 1
    999?
whales_tx[c(2, 10)]
whales_tx[c("1991", "1999")]
whales_tx[c(F, T, F, F, F, F, F, F, F, T)]
```

Common Functions

- Common functions for a numeric vector
  - `sum(`x`)` and `mean(`x`)`
  - `median(`x`)`
  - `min(`x`)`, `max(`x`)`, and `range(`x`)`
  - `sort(`x`)` and `sort(`x`, decreasing = TRUE)`
  - `rank(`x`)`
  - `rev(`x`)`
  - `which.min(`x`)` and `which.max(`x`)`
- Mathematical operators and functions are applied entry-wise

Common Functions

- Examples

```
mean(whales_tx)

median(whales_tx)

range(whales_tx)
sort(whales_tx)
sort(whales_tx, decreasing = TRUE)
which.max(whales_tx)
which.min(whales_tx)

log(whales_tx)
```

Common Functions

- Examples

```
# Input the number of whales beachings per
    year in Texas during the 1990s
whales_tx <- c(74, 122, 235, 111, 292, 111,
    211, 133, 156, 79)
names(whales_tx) <- c(1990, 1991, 1992, 1993
    , 1994, 1995, 1996, 1997, 1998, 1999)

# Input the number of whales beachings per
    year in Florida during the 1990s
whales_fl <- c(89, 254, 306, 292, 274, 233,
    294, 204, 204, 90)
names(whales_fl) <- names(whales_tx)

# What is their difference
whales_diff <- whales_fl - whales_tx
```

Common Functions

- Subset a vector
  - For a numeric vector: x`[which(`$x \odot c$`)]` or x`[`$x \odot c$`]`, where $\odot \in \{$`==`, `!=`, `>=`, `>`, `<=`, `<`$\}$ and c is a desired number

Common Functions

- Examples

```
# Which years (indices) the number of whales
    in Texas was equal or greater than 200?
which(whales_tx >= 200)

# What are the numbers of whales in Texas in
    those years?
whales_tx[which(whales_tx >= 200)]
whales_tx[whales_tx >= 200]

#  Which years (indices) the number of
   whales in Texas was greater than 100 and
   equal or less than 200?
which(whales_tx > 100 & whales_tx <= 200)
```

Common Functions

- Examples

```
# Which years (indices) the number of whales
    in Texas was greater than the one in
   Florida?
indices <- which(whales_tx > whales_fl)

# What are the numbers of whales in Texas in
    those years?
whales_tx[indices]
whales_tx[which(whales_tx > whales_fl)]

# What are the numbers of whales in Florida
   in those years?
whales_fl[indices]
whales_fl[which(whales_tx > whales_fl)]
```

Common Functions

- Other functions
  - `length(`x`)`
  - `mode(`x`)`: numeric, character, logical, and complex
  - `is.numeric(`x`)`, `is.integer(`x`)`, `is.character(`x`)`, `is.logical(`x`)`
  - `as.numeric(`x`)`, `as.integer(`x`)`, `as.character(`x`)`, `as.logical(`x`)`
  - `rm(`x`)`
  - `unique(`x`)`
  - `c(`x`, `y`)`

Your Turn

- You track your commute times for one week (five weekdays), recording the following times in minutes:

$$17, 16, 20, 24, 18.$$

  Enter the data into a numeric vector, do the following things:

  - Enter this data into a variable called commute
  - Name the data vector with weekday abbreviations
  - What is the total commute time for this week?
  - Using `diff()`, find the day with the greatest change from the previous day

Your Turn

- Solutions

```
# Enter data
commute <- c(17, 16, 20, 24, 18)

# Name data
names(commute) <- c("Mon", "Tue", "Wed", "
    Thu", "Fri")

# Find the total commute time
print(sum(commute))

# Find the day with greatest change
commute_diff <- diff(commute)
commute_diff_abs <- abs(commute_diff)
print(names(which.max(commute_diff_abs)))
```

Examples

- Examples

```
# Input the number of whales beachings per
    year in Texas during the 1990s
whales_tx <- c(74, 122, 235, 111, 292, 111,
    211, 133, 156, 79)
names(whales_tx) <- c(1990, 1991, 1992, 1993
    , 1994, 1995, 1996, 1997, 1998, 1999)

# An improved way
names(whales_tx) <- 1990:1999
```

Simple Sequences

- The data vector that has certain structure or pattern
- Simple sequence
  - Numeric:

$$x_1, \ldots, x_i, \ldots, x_n, \text{ where } x_1 \in \mathbb{R} \text{ and } x_i = x_1 + (i-1)$$

    , via the operator $x_1 : x_n$
  - Character:

$$a, \ldots, z \text{ or } A, \ldots, Z$$

    , via the built-in variables `letters` or `LETTERS`

Arithmetic Sequences

- Arithmetic sequence type I:

$$x_1, \ldots, x_i, \ldots, x_n, \text{ where } x_1 \in \mathbb{R} \text{ and } x_i = x_1 + (i-1)h$$

  - x_1 is the starting point and h is the step size
  - Realized via the function `seq(`x_1`, `x_n`, by = h)`

- Arithmetic sequence type II:

$$x_1, \ldots, x_i, \ldots, x_n, \text{ where } x_1 \in \mathbb{R} \text{ and } x_i = x_1 + \frac{(x_n - x_1)i}{n-1}$$

  - x_1 is the starting point and x_n is the ending point
  - Realized via the function `seq(`x_1`, `x_n`, length.out = n)`

Repeated Sequences

- Suppose the repeating unit is $x_1, \ldots, x_K$
- Repeated sequences type I:

$$\overbrace{x_1, \ldots, x_1}^{m}, \ldots, \overbrace{x_k, \ldots, x_k}^{m}, \ldots, \overbrace{x_K, \ldots, x_K}^{m}, \text{ where } n = mK$$

  - Realized via the function `rep(c(`x_1`,...,`x_K`), each = m)`
- Repeated sequences type II:

$$\overbrace{x_1, \ldots, x_K, x_1, \ldots, x_K, x_1, \ldots, x_K}^{m}, \text{ where } n = mK$$

  - Realized via the function `rep(c(`x_1`,...,`x_K`), times = m)`

Structured Data Vector

- Examples

```
# Simple sequence
n <- 10
0:(n - 1)
letters

# Repeated sequences
rep(1:10, each = 2)
rep(1:10, 2)
rep(1:10, times = 2)

# Arithmetic sequence
seq(1, 10, by = 2)
seq(1, 10, length.out = 5)
1 + 2 * (0:4)
```

Structured Data Vector

- Examples

```
names(whales_tx) <- 1990:1999
names(whales_fl) <- 1990:1999

# What is the numbers of whales per odd
    number year in Texas
n <- length(whales_tx)
whales_tx[seq(2, n, by = 2)]
```

Your Turn

- Create the following sequences by using `:`, `seq()`, and `rep()`
  - "a", "a", "a", "a", "a", "a"
  - All the years between your birth year and this year
  - $1, 3, \ldots, 99$
  - $1, 1/2, 1/3, 1/4, 1/5, 1/6, 1/7, 1/8, 1/9, 1/10$
  - $1, 8, 27, 64, 125, 216$
  - $0, 25, 50, 75, \ldots, 1000$

Your Turn

- Solutions

```
# Question 1
rep("a", 6)

# Question 2
2000:2019

# Question 3
seq(1, 99, by = 2)
seq(1, 99, length.out = 50)
2 * (1:50) - 1
```

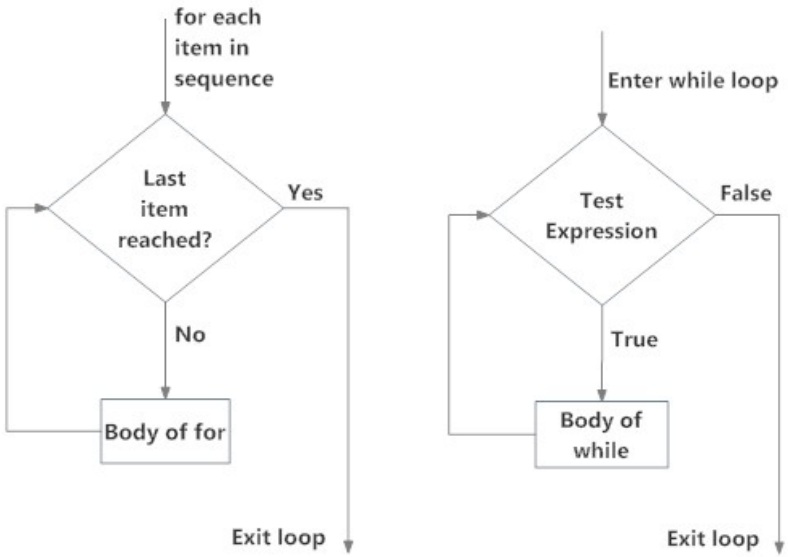
Your Turn

- Solutions

```
# Quesiton 4
1/(1:10)

# Question 5
(1:6)^3

# Question 6
seq(0, 1000, by = 25)
```

Loop Statements

- A way to repeat a sequence of commands
- Allows to automate parts of code that need repetition
- Three basic forms:
  - For: `for(){}`, fed by a vector, i.e. a structured numeric vector
  - While: `while(){}`, fed by a logical value
  - Repeat: `repeat{{} if(){break}}`, fed by a logical value



Loop Statements

- Examples

```
s <- 1 + 2 + 3 + 4 + 5

s <- 0
for (i in 1:5) {
  s <- s + 1
}
print(s)
```

Loop Statements

- Examples

```
whales_tx <- c(74, 122, 235, 111, 292, 111,
    211, 133, 156, 79)
whales_fl <- c(89, 254, 306, 292, 274, 233,
    294, 204, 204, 90)

# What is the number of whales beachings per
    year in both Texas and Florida?
whales <- whales_tx + whales_fl
```

Loop Statements

- Examples

```
# For loop
n <- length(whales_tx)
whales <- rep(0, n)
for (i in 1:n) {
  whales[i] <- whales_tx[i] + whales_fl[i];
}

# While loop
i <- 1;
while (i <= n) {
  whales[i] <- whales_tx[i] + whales_fl[i];
  i <- i + 1;
}
```

Loop Statements

- Sum all even numbers in a vector

```
x <- c(2, 5, 3, 9, 8, 11, 6)
count <- 0
for (i in x) {
  if(i %% 2 == 0) {
    count <- count + 1
  }
}

# Alternative
for (i in 1:length(x)) {
  if(x[i] %% 2 == 0) {
    count <- count + 1
  }
}
```

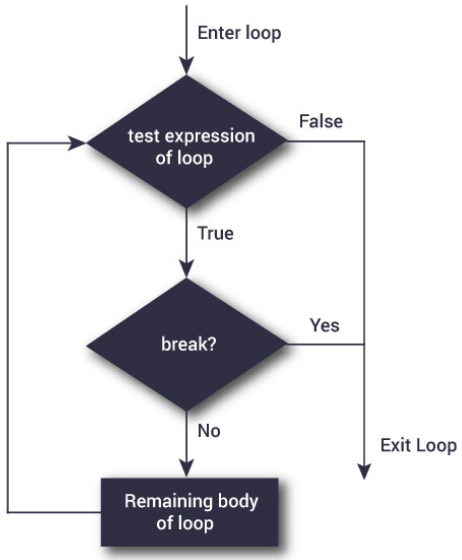
Loop Statements

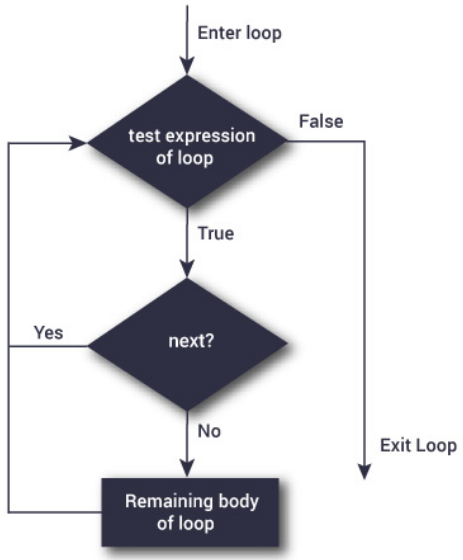
- Insert a loop within another loop

```
# Print all outcomes of tossing two dices
for (i in 1:6) {
  for (j in i:6) {
    print(i + j)
  }
}
```

Loop Statements

- Alter a loop by
  - `break`: stop the iterations and flow the control outside of the loop
  - `next`: skip the current iteration of a loop without terminating it





Your Turn

- A prime number is an integer greater than one whose only factors are one and itself. For example, the first ten prime numbers are $2, 3, 5, 7, 11, 13, 17, 19, 23$ and 29. Write a function that returns all prime numbers between 1 and a given number n.

Your Turn

- Break complex tasks into smaller parts
- Use words to describe how each part should work
- Translate words to R by using
  - Condition statements
  - Loop statements
  - Functions
- When all parts work, combine into a function

Your Turn

- Use plain language to describe how to obtain the number of prime numbers in the range of $[1, n]$
  - List all the integers between 1 and n
  - Try if each one, say i, is divisible by any other integers between 2 and $i - 1$
  - If i is non-divisible, then it is a prime number
  - List all the prime numbers between 1 and n

Your Turn

- Translate the plain language into R

```
is.prime <- rep(TRUE, n)
is.prime[1] <- FALSE
if (n > 2) {
  for (i in 3 : n) {
    for (j in 2 : (i - 1)) {
      if (i %% j == 0) {
        is.prime[i] <- FALSE
        break
      }
    }
  }
}
primes <- which(is.prime)
```

Your Turn

- Wrap up a function

```
list.primes = function(n) {
  is.prime <- rep(TRUE, n)
  is.prime[1] <- FALSE
  if (n > 2) {
    for (i in 3 : n) {
      is.prime[i] <- TRUE;
        for (j in 2 : (i - 1)) {
          if (i %% j == 0) {
            is.prime[i] <- FALSE
            break
          }
        }
    }
  }
  return (which(is.prime))}
```

Your Turn

- Handle the invalid input

```
list.primes = function(n)
  if (length(n) > 1) {
    error("The input is multiple!")
  } else if (!is.numeric(n)) {
    error("The input is a non numeric!")
  } else if (n < 2) {
    error("The integer is less than 2!")
  } else {
    ...
  }
  ...
}
```

The Brackets

- Parentheses (i.e. round brackets): ()
  - Encompass input arguments of a function
  - Overwrite the priority rule of arithmetic/logical operators
- Square brackets: []
  - Access entries in a vector, a matrix, or a data frame
- Curly braces (i.e. curly brackets): {}
  - Encompass the body of a self-written function, a if statement, or a loop statement

Revisit "Your Turn"

- Fibonacci number: Each number is the sum of the two preceding ones, starting from 0 and 1
- Use `R` to get the i-th Fibonacci number

Revisit "Your Turn"

- Recursion

```
Fibo = function(i) {
  if (i <= 2) {
    return (i - 1)
  } else {
    return (Fibo(i - 1) + Fibo(i - 2))
  }
}
```

Your Turn

- Get the i-th Fibonacci number using a solution with low time complexity
- Create a Fibonacci sequence with length 100 and access the i-th number

Your Turn

- Solutions

```
Fibo_seq <- as.numeric(0:99)
for (i in 3:100) {
  Fibo_seq[i] <- Fibo_seq[i - 1] + Fibo_seq[
      i - 2]
}
```

- Comparison

```
start_time <- proc.time()
Fibo(35)
end_time <- proc.time()
print(end_time - start_time)

Fibo_seq[35]
```

After-class Reading

- *Using R for Introductory Statistics (1st Ed.)* by John Verzani
- Chapter 1 Data
  - Section 1.2 Some `R` essentials
    - Subsection 1.2.3 Assignment
    - Subsection 1.2.4 Using `c()` to enter data
    - Subsection 1.2.5 Using functions on a data vector
    - Subsection 1.2.6 Creating structured data
  - Section 1.3 Accessing data by using indices
    - Subsection 1.3.1 Assigning values to data vector
    - Subsection 1.3.2 Logical values
    - Subsection 1.3.4 Managing the work environment