Northeastern University

CS6020: Collecting, Storing, and Retrieving Information

Programming in R

Programming in R

BASIC R PROGRAMMING

Lesson Objectives

- After completing this lesson, you are able to:
  - manipulate objects
  - create data frames
  - organize code with functions
  - perform logical queries on data sets
  - detect and eliminate missing values
  - calculate summary statistics
  - practice with built-in data sets

R Objects

- Data is stored as objects in R.
- Objects are created by:
  - Reading data from an external file
  - Retrieving data from a URL
  - Creating an object directly from the command line
  - Instantiating an object from within a program

Creating R Objects Directly

```
> x <- 99

> x
[1]99

> a <- "hello"
> a
[1]"hello"
```

Note that object names are case sensitive and cannot contain spaces or special characters. An object identifier must start with a letter, but may contain any letter or digit thereafter.

Case Sensitivity

- Note that R is case sensitive which means that R treats the object names `AP` and `ap` as different objects.
- Accessing files is also most commonly case sensitive, so there's a difference between "*AirPassengers.txt*" and "*airpassengers.txt*".

Organizing Code with Functions

- Functions are an important part of programming in R.
- They allow code to be reused.

```
> fraction<-function(x,y){
+ result <- x/y
+ print (result)
+ }
> fraction(3,2)
[1] 1.5
```

Concatenation and Arrays

- Concatenating numeric or character values using the built-in `c()` function results in an indexable array.

```
> a <- c(1,2,3,4,5)
> a
[1] 1 2 3 4 5
> a + 10
[1] 11 12 13 14 15
> a
[1] 1 2 3 4 5
> b <- a/2
> b
[1] 0.5 1.0 1.5 2.0 2.5
> c <- a + b
> c
[1] 1.5 3.0 4.5 6.0 7.5
```

Listing and Deleting Objects

- To view all of the objects in current R session:

```
ls()
[1] "a"          "b"          "c"          "fraction" "x"
```

- To remove an object, use the `rm()` function:

```
rm (a)
> a
Error: object 'a' not found
```

Deleting all Objects

- To irretrievably remove all objects from current session:

```
> rm (list=ls())
```

Comments

- Longer scripts should be commented so that you or others understand the intent of the commands and functions.
- Any text after a hash mark (#) is ignored by R.

```
> # create new object with initial value 25
> a <- 25
```

Summary

- In this lesson, you learned that:
  - in R variables are called objects
  - object names are case sensitive
  - functions help organize reusable code
  - comments help explain a script

Sequences and Subscripting

- Numeric sequences are an important programming mechanism in R.

```
> # create new object with initial value 25
> a <- 25
> 1:10
 [1]  1  2  3  4  5  6  7  8  9 10
> 5:12
[1]  5  6  7  8  9 10 11 12
> 3:-3
[1]  3  2  1  0 -1 -2 -3
> 2*1:5
[1]  2  4  6  8 10
> 2*(1:5)
[1]  2  4  6  8 10
```

Sequences with `seq()`

- The seq() function is used to generate more elaborate sequences.

```
> seq(from=5,to=15,by=3)
[1]  5  8 11 14
> seq(from=1,to=10,length=6)
[1]  1.0  2.8  4.6  6.4  8.2 10.0
> seq(from=100,length=4,by=-2.5)
[1] 100.0  97.5  95.0  92.5
> x <- 10:20
> seq(from=50,to=52,along=x)
 [1] 50.0 50.2 50.4 50.6 50.8 51.0 51.2 51.4
51.6 51.8 52.0
```

Indexing/Subscripting

- Sequences are essentially arrays and particular elements of the sequence can be extracted with the `[]` subscript operator.
- Subscripting in R is much more flexible than many other programming languages.

```
> # extract the 3rd element
> x[3]
[1] 12
> # extract all BUT the 3rd element
> x[-3]
 [1] 10 11 13 14 15 16 17 18 19 20
```

Retrieving Selected Elements

- Concatenation can be used in conjunction with sequencing to retrieve a subset of elements.

```
> x
 [1] 10 11 12 13 14 15 16 17 18 19 20
> #retrieve the 5th and 7th elements
> x[c(5,7)]
[1] 14 16
> #retrieve all but the 3rd, 5th, and 9th elements
> x[c(-3,-5,-9)]
[1] 10 11 13 15 16 17 19 20
```

Filtering with Subscripting

- Specific elements meeting a logical criterion can be selected using subscripting.

```
> x
 [1] 10 11 12 13 14 15 16 17 18 19 20
> #extract all elements greater than 14
> x[x>14]
[1] 15 16 17 18 19 20
```

Data Frames

- Data frames are the most common type of compound data structure used in R in addition to scalar values (vectors) and collections of values (array sequences).
- They are similar to C++ and Java objects or C `struct`'s.
- A data frame is composed of multiple values each of which is commonly a sequence.

Data Frames

- A data frame is often created by loading data from an external file or created internally.
- Data frames are essentially spreadsheets of columns and rows.

```
> x<-1:10
> y<-seq(from=100,to=300,by=5)
> # create a new data frame 'df'
> df <- data.frame(x,y)
Error in data.frame(x, y) :
  arguments imply differing number of rows: 10, 41
> y<-seq(from=100,to=300,length=10)
> df <- data.frame(x,y)
```

Accessing Elements of a Data Frame

```
> df[4,2]
[1] 166.6667
```

```
> df
    x        y
1   1 100.0000
2   2 122.2222
3   3 144.4444
4   4 166.6667
5   5 188.8889
6   6 211.1111
7   7 233.3333
8   8 255.5556
9   9 277.7778
10 10 300.0000
```

The above reference extracts the element in row 4, column 2.

More Data Frame Accesses

```
> df[2]
          y
1  100.0000
2  122.2222
3  144.4444
4  166.6667
5  188.8889
6  211.1111
7  233.3333
8  255.5556
9  277.7778
10 300.0000
```

```
> df[,2]
 [1] 100.0000 122.2222 144.4444
166.6667 188.8889 211.1111 233.3333
255.5556
 [9] 277.7778 300.0000
> df[7,]
  x        y
7 7 233.3333
> df$x
 [1]  1  2  3  4  5  6  7  8  9 10
```

Object Dimensions

- The number of rows or columns of a data frame can be queried.

```
> ncol(df)
[1] 2
> nrow(df)
[1] 10
> dim(df)
[1] 10  2
> dim(df)
[1] 10  2
> length(df$x)
[1] 10
```

Referencing the Final Element

- To get to the last element in an array, use this technique:

```
> x
 [1]  1  2  3  4  5  6  7  8  9
10
> x[length(x)]
[1] 10
```

Practicing with Built-In Data

- R has several built-in data sets that can be used to “practice”.
- The “sunspot” data set contain annual observed sunspot activity from 1799 to 1988.

```
> sunspot.year
Time Series:
Start = 1700
End = 1988
Frequency = 1
  [1]   5.0  11.0  16.0  23.0  36.0  58.0  29.0
20.0  10.0   8.0   3.0   0.0
 [13]   0.0   2.0  11.0  ...
```

Converting Data to a Frame

- To convert this data set into tabular format, use the `data.frame` function.

```
> sunspots <-
data.frame(year=1700:1988,count=sunspot.year)
> head(sunspots)
  year count
1 1700     5
2 1701    11
3 1702    16
4 1703    23
5 1704    36
6 1705    58
```

The `head()` and `tail()` Functions

- The `head()` and `tail()` functions list the first or last six rows of a data frame.

```
> head(sunspots)
  year count
1 1700     5
2 1701    11
3 1702    16
4 1703    23
5 1704    36
6 1705    58
```

```
> tail(sunspots)
    year count
284 1983  66.6
285 1984  45.9
286 1985  17.9
287 1986  13.4
288 1987  29.2
289 1988 100.2
```

Rounding Decimal Values

- Often decimal values are needed as integers.
- This can be accomplished with the `round()` function.

```
> sunspots_int<-round(sunspots)
> head(sunspots_int)
  year count
1 1700     5
2 1701    11
3 1702    16
4 1703    23
5 1704    36
6 1705    58
```

Testing Set Inclusion

- Given a logical statement, `any()` tests if at least one value in the set meets the criterion.

```
> any(sunspots_int[,2] < 0)
[1] FALSE
> any(sunspots_int[,1] < 1700 | sunspots_int[,1] > 1988)
[1] FALSE
```

Descriptive Statistics

```
> mean(sunspots_int[,2])
[1] 48.63322
> round(mean(sunspots_int[,2]))
[1] 49
> round(mean(sunspots_int[,2]),digits=2)
[1] 48.63
> max(sunspots_int[,2])
[1] 190
> which(sunspots_int[,2] == 190)
[1] 258
> sunspots_int[258,]
    year count
258 1957   190
```

Queries on Data Frames

- Suppose we need to know:

  "How many years were fewer than five sunspots observed?"

```
> length(which(sunspots_int[,2] < 5))
[1] 16
```

Queries on Data Frames

- Suppose we need to know:

  "In which years were fewer than five sunspots observed?"

```
> sunspots_int[(which(sunspots_int[,2] < 5)),]
    year count
11  1710     3
12  1711     0
13  1712     0
14  1713     2
99  1798     4
…
```

Summary Statistics

- To obtain quick summary statistics on a data object, use the `summary()` function.

```
> summary(sunspots_int[,2])
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
   0.00   16.00   39.00   48.63   69.00  190.00 
```

Summing Columns

- To sum a column in a data frame, use the `colSums()` function.

```
> colSums(sunspots_int[2])
count
14055
```

- Note that the `colSums()` function requires an array reference rather than a data frame, therefore no comma.

Missing Values

- Missing data values in a data frame are encoded as `NA`.
- A missing value bars any calculation of summary statistics or numeric expression.

Air Quality Data Set

- The built-in data set "*airquality*" contains measurements of daily air quality in New York City from May through September 1973.

```
> head(airquality)
  Ozone Solar.R Wind Temp Month Day
1    41     190  7.4   67     5   1
2    36     118  8.0   72     5   2
3    12     149 12.6   74     5   3
4    18     313 11.5   62     5   4
5    NA      NA 14.3   56     5   5
6    28      NA 14.9   66     5   6
> mean(airquality$Solar.R)
[1] NA
```

Handling Missing Values

- If the values cannot be "extrapolated" then they should be removed from any calculation.

```
> any(is.na(airquality))
[1] TRUE
> mean(airquality$Solar.R,na.rm=TRUE)
[1] 185.9315
> which(is.na(airquality$Solar.R))
[1]  5  6 11 27 96 97 98
```

Removing Missing Values

- Missing values can be removed from the data set using the `na.omit()` function.

```
> air_complete <- na.omit(airquality)
> head(air_complete)
  Ozone Solar.R Wind Temp Month Day
1    41     190  7.4   67     5   1
2    36     118  8.0   72     5   2
3    12     149 12.6   74     5   3
4    18     313 11.5   62     5   4
7    23     299  8.6   65     5   7
8    19      99 13.8   59     5   8
```

Saving Script Files

- R commands can be created in a text file and loaded on demand rather than typing it in over and over.
- Create a text file in a text editor and save the file with the `.R` extension.
- Use the `source()` function to load and execute the script.

Example Source File

```
# Simple R script: created.R
x<-1:10
y<-seq(from=100,to=300,length=10)
df <- data.frame(x,y)
```

```
> source("createDF.R")
> df
    x        y
1   1 100.0000
2   2 122.2222
3   3 144.4444
4   4 166.6667
5   5 188.8889
6   6 211.1111
7   7 233.3333
8   8 255.5556
9   9 277.7778
10 10 300.0000
```

Getting Help with `help()`

- To get help for any function use the `help()` function.

```
> help(seq)
```

seq {base} R Documentation

Sequence Generation

Description

Generate regular sequences. `seq` is a standard generic with a default method. `seq.int` is a primitive which can be much faster but has a few restrictions. `seq_along` and `seq_len` are very fast primitives for two common cases.

Usage

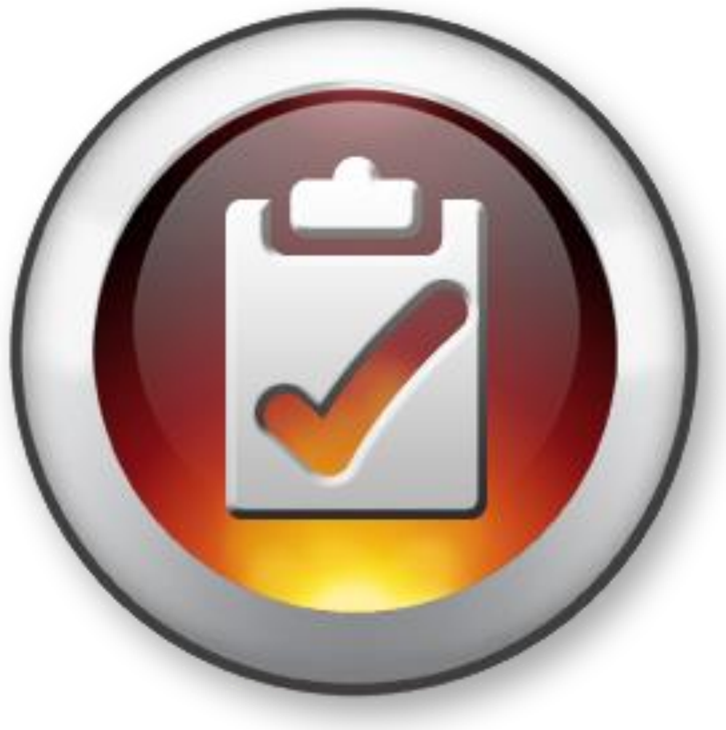
```
seq(...)

## Default S3 method:
seq(from = 1, to = 1, by = ((to - from)/(length.out - 1)),
    length.out = NULL, along.with = NULL, ...)

seq.int(from, to, by, length.out, along.with, ...)

seq_along(along.with)
seq_len(length.out)
```

Arguments

`...`	arguments passed to or from methods.
`from, to`	the starting and (maximal) end values of the sequence.

Summary

- In this lesson, you learned that:
  - R contains several built-in data sets
  - data frames are one of the most fundamental object types in R
  - functions help organize reusable code
  - logical query functions which() and any()help identify key values
  - missing data is encoded with NA but must be eliminated before calculations can be performed



Summary, Review, & Questions…