DATA 133 - Introduction to Data Science I

Instructor: Renzhi Cao
Computer Science Department
Pacific Lutheran University

For the pair practice, Question #7: You should print the #4798 row.

Announcements

- Read books: Page 70-78
- Pair Practice from last Thursday is due, please submit on Sakai.
- **Thursday class is canceled**, but can meet on Discord, review and pair practice.
- **Please check Week 7 course recording on Sakai!**

Announcements

- Review Pair Practice from previous Thursday, do it together. Solution will also be posted on course website.
- Quiz 3 on Sakai (do it now or later on before 10 pm?)

Break

Reference book

- R Programming for Data Science. By Roger Peng.
  ISBN-10: 1365056821, April 20, 2016.

Pictures from: https://www.google.com/search?

Learning in today

- R basics - Function.

What is function?

- A large program in R can be divided to many subprogram
- The subprogram passes a self contain components and have well define purpose.
- The subprogram is called as a function.
- Function - do a task.

What is function?

Why we need function?

- It will be much easier to divide a big task into several smaller and simpler tasks.
- Allowing the code to be called many times
- Easier to read and update
- Easier to debug R program, find and fix errors

What is function?

- Writing functions is a core activity of an R programmer.
- Functions in R are “first class objects”, which means that they can be treated much like any other R object.
- Functions can be passed as arguments to other functions.
- Functions can be nested, so that you can define a function inside of another function.

First R function

```
> f <- function() {
+ ## This is an empty function
+}

> ## Functions have their own class
> class(f)
[1] "function"
> ## Execute this function
> f()
NULL
```

Not very interesting, but it's a start.

Demo how to write a function and save it to a file, use "source" to load the function.

Practice

- Create a function f, add statement to the function: print("Hello World"). Save it to a file called myFunction.R.

- Use source to load the file, and call function f.

How the function works

- R program doesn't execute the statement in function until the function is called.
- When the function is used it is referred to as the **called function**.
- Data is passed from a R program/function to a called function by specifying the variables in a argument list.

How the function works

```
> f <- function(n) {
+   print("Hi")
+.  print(n)
+}
```

What will the program print?

```
> f (3)
```

What will the program print?

```
> for(i in 1:3) {
    f (i)
}
```

What will the program print?

How the function works

```
>f <- function(num){
    for(i in seq_len(num)) {
        print("Hello, world!\n")
    }
    }
>f(3)
```

What will the program print?

```
>f <- function(n){
    for(i in seq_len(n)) {
        print("Hello, world!\n")
    }
    }
>f(3)
```

What will the program print?

How the function works

- The above function doesn't return anything.
- It is often useful if a function returns something that might be fed into another section of code.

```
>f <- function(num){                This function returns the total number
   Hello <- "Hello world!\n"         of characters printed to the console
   for(i in seq_len(num)) {
       cat(Hello)
   }
    Chars <- nchar(Hello) * num
    Chars
   }
>f(3)
```

```
>meaningoflife <- f(3)          # what will print?
>print(meaningoflife)           # what will print?
>f()                            # what happens?
```

How the function works

```
>f <- function(num = 3){
    Hello <- “Hello world!\n”
    for(i in seq_len(num)) {
        cat(Hello)
    }
     Chars <- nchar(Hello) * num
     Chars
    }
>f(3)
```

This function returns the total number of characters printed to the console

```
>f()
```

what happens?

How the function works

```
>x <- 0
>f <- function(num){
        x <- 10
        x+num
   }
>y <- f(3)
>x
```

What will the program print?

Argument matching

R functions arguments can be matched positionally or by name. Positional matching just means that R assigns the first value to the first argument, the second value to second argument, etc.

Let's check the example of rnorm function.
>str(rnorm) # you can also use ?rnorm to understand more about rnorm

Positional match first argument, default for 'na.rm'
>mydata <- rnorm(100, 2, 1) *## Generate some data*

>str(sd)
>sd(mydata)
>sd(x=mydata)
>sd(na.rm=FALSE, x = mydata) ## specified both arguments by name

Practice

- Create a function f with two parameters p1 and p2, return the summation of p1 and p2. Test your function by calling:

```
>sum <-f(2,3)
>print(sum)
```

- Write a function f2 with one parameter m, display values from 1 to m. Test your function by calling:

```
>f2(50)
```

- Write a function f3 with one parameter n, display a n*n square of *. Test your function by calling (Use cat instead of print if you don't want new line):

```
>f3(4)
# you should get:
* * * *
* * * *
* * * *
* * * *
```

END

- Finish the quiz by 10 pm today and work on the pair practice posted on Sakai!
- The first course project has been posted on Sakai as well.
- No class this Thursday, **check Sakai Week 7 recording**, but can meet 1-on-1 on Discord, review and work on the exercise and course project 1!

Useful statistics function

Function	Description
mean(*x*, trim=0, na.rm=FALSE)	mean of object x # trimmed mean, removing any missing values and # 5 percent of highest and lowest scores mx <- mean(x,trim=.05,na.rm=TRUE)
sd(*x*)	standard deviation of object(x). also look at var(x) for variance and mad(x) for median absolute deviation.
median(*x*)	median
quantile(*x*, *probs*)	quantiles where x is the numeric vector whose quantiles are desired and probs is a numeric vector with probabilities in [0,1]. # 30th and 84th percentiles of x y <- quantile(x, c(.3,.84))
range(*x*)	range
sum(*x*)	sum
diff(*x*, lag=*1*)	lagged differences, with lag indicating which lag to use
min(*x*)	minimum
max(*x*)	maximum
scale(*x*, center=TRUE, scale=TRUE)	column center or standardize a matrix.

Useful statistics function

Function	Description
seq(*from* , *to*, *by*)	generate a sequence indices <- seq(1,10,2) #indices is c(1, 3, 5, 7, 9)
rep(*x*, *ntimes*)	repeat *x* *n* times y <- rep(1:3, 2) # y is c(1, 2, 3, 1, 2, 3)

Cor(x, y = NULL, use = "everything", method = c("pearson", "kendall", "spearman"))

Calculate correlation between two vectors.

Useful statistics function

R has some functions which implement looping in a compact form to make your life easier.

lapply(): Loop over a list and evaluate a function on each element:
>str(lapply)

example
>mylist <- list(a=1:10, b=20:100, c=30:50)
>lapply(mylist,mean)

Practice

- Use seq and rep function. First create vector v1 with odd numbers from 0 to 100. And then create vector v2 which repeats the vector v1 three times.

- Calculate the mean, standard deviation, median, sum, min, max and range of v3.

- Create two vectors: (1,2,3,4,5,6), (9,8,7,6,5,4), use Cor function to calculate the correlation between this two vectors. (This might be useful for your project).

Practice of R

Try it on course website!
Pair-programming is due at the end of next class (We will give you time to do the pair practice on Thursday)
Read the book