CS 133 - Introduction to Computational and Data Science

Instructor: Renzhi Cao
Computer Science Department
Pacific Lutheran University
Spring 2017

Announcement

- *Read book to page 78.*
- *Final project*
- *Today we are going to learn R control structure and function.*

For loop

In the example of calculating summation of all elements in a vector:

```
v <- c(10, 20, 30)

sumOfV <- v[1] + v[2] + v[3]
```

What happens when v has 100 elements?

You need to have a loop to do that!

For loops are pretty much the only looping construct that you will need

in R.

```
for(<condition>) {
## repeat doing something until condition is false
}
```

For loop

```
> for(i in 1:10)
  {
      print(i)
  }
> x <- c("a", "b", "c", "d")
> for(i in 1:4) {
      +  ## Print out each element of 'x'
      +          print(x[i])
      +}
```

For loop

The seq_along() function is commonly used in conjunction with for loops in order to generate an integer sequence based on the length of an object (in this case, the object x).

```
> ## Generate a sequence based on length of 'x'
> for(i in seq_along(x)) {
+         print(x[i])
+}

## alternative
> for(i in seq_along(x)) print(x[i])
```

Exercises

- Create a R code file: PracticeR2.R, and save today's code in that file
- Create a vector y <- c(1,2)
- Set the third element of y as 3
- Use for loop to set i element of y as i. (i from 4 to 20)
- Use seq_along to print each element of y

Nested For loop

The loops can be nested inside each other.

```
x <- matrix(1:6, 2, 3)
for(i in seq_len(nrow(x))) {
for(j in seq_len(ncol(x))) {
        print(x[i, j])
}
}
```

- seq_len(integer i) — return [1,2, …, i]
- seq_along(vector or list?) — return [1,2,…, length of the vector or list]

Nested For loop

Hints for your final project:

If you get all data in data frame d, you can use the following statement to do analysis between every feature pairs.

```
for(i in seq_len(nrow(d)) {
for(j in seq_len(ncol(d)) {
    if(i!=j)
    {
        f1 <- d[,i]
        f2 <- d[,j]
        # now analysis these two features f1 and f2 ….
    }
}
}
```

While loop

While loops begin by testing a condition. If it is true, then they execute the loop body. Once the loop body is executed, the condition is tested again, and so forth, until the condition is false, after which the loop exits.

```
> count <- 0
> while(count < 10) {
+         print(count)
+  count <- count + 1
+}
```

next and break

- next is used to skip an iteration of a loop.
- break is used to exit a loop immediately, regardless of what iteration the loop may be on.

```
for(i in 1:100) { if(i <= 20) {
        ## Skip the first 20 iterations
      next
}
    print(i)
}
```

```
for(i in seq(1,100,1)) { if(i > 20) {
        ## stop at 20 iterations
      break
}
    print(i)
}
```

Practice

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

```
for(i in seq_along(v)) {
if(i >2) {
break
}
print(i)
}
```

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

```
for(i in seq_along(v)) {
if(i <=2) {
next
}
print(i)
}
```

Exercises

- Create two matrix m1 and m2 as follows:

m1: 1 3 m2: 5 7

2 4 6 8

- Create a 2*2 matrix m3, which is element wise multiplication of m1 and m2. Use for loop to calculate the value of m3. The value of m3 should be:

m3: 5 21

12 32

Solution

```
m1<-matrix(1:4,2,2)

m2<-matrix(5:8,2,2)

m3<-matrix(nrow=2,ncol=2)

for(i in seq_len(nrow(m1))){

    for(j in seq_len(ncol(m1))){

        m3[i,j] = m1[i,j] * m2[i,j]

    }

}
```

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.

Functions

- 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

Functions

- 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.

Exercises

- Continue to work on PracticeR2.R. Create a function f, add statement to the function: print(“Hello World”)

- Use source to load the function 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?

Called function, and data 3 is passed to the function.

```
> 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?
```

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
```

Exercises

- 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:

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

Exercises

- For a function f10, write a for loop to display the values from 1 to 25 along with each value squared. The output should look like this:

1 squared is 1

2 squared is 4

3 squared is 9

- For a function f11, write a for loop to print the odd numbers from 1 to 99 (inclusive). Hint: i%2 == 0 means i is odd number, so you may use if statement also.
- For a function f12, Write a for loop to display the multiples of 3 from 99 down to 3.