R programming for beginners

Ni Shuai

Computational Genome Biology
German Cancer Research Center (DKFZ)

November, 2016

Control Flow

There are 3 types of basic control-flow constructs:

- If some condition is TRUE , do something, if its FALSE, do something else
  - if (condition) {do}
  - if (condition) {do} else {do}
  - ifelse(condition, yes, no)
- Run the same command for a fixed number of times
  - for (vales in sequence) {do}
- Run the same command while a condition is TRUE
  - while (condition) {do}
  - repeat {do; if (condition) break}

Meet the condition

The 'if-then-else' statement comes handy when we need to perform different actions depending on weather a condition is true or false, its basic structure looks like this:

```
Time=Sys.time() #Get the system's current time
Time=format(Time, '%H') #Extract Decimal hour (24 hour)

if (Time < 12 & Time > 8) {
    print('Guten morgen!')
} else {
    print('Guten tag!')
}

## [1] "Guten tag!"
```

We extracted the the current time and tought R to greet us smartly. If **Time < 12 & Time > 8** is true, the expression **print('Guten morgen!')** will be excuted, otherwise the expression **print('Guten tag!')** will be excuted.

Meet the condition — Tips

- The condition must be of a single boolean value, that is, either TRUE or FALSE
- The 'else' part can be omitted
- If there is an 'else' statement, the 'else' part must continue right after where the 'if' part ends (else will never start from a new line)
- If the expressions is only one line of code, brackets can be omitted, but its wise to always have them for its maintainbility

Exercise:

- Given an unknown real number variable a, return the absolute value of a
- Given a numeric vector of length 2, find out the maximum out of the two numbers

vectorized if-then-else in R

Similar to the if-else statement, R has a vectorized if-else which takes an expression as a logic vector and returns a result vector of the same length, it takes the form

ifelse(Expression, x, y)

For each position in Expression, the returned vector takes x if the expression in that position is TRUE, otherwise it takes y.
for example:

```
numbers=1:8
ifelse(numbers %% 2 ==0, 'Even','Odd')

## [1] "Odd"  "Even" "Odd"  "Even" "Odd"  "Even" "Odd"  "Even"
```

Meet multiple conditions

The if-else statement allows us to repond diffently to two alternatives, but sometimes things get more complicated, in case we have more than two conditions to choose from, we can use nested if-else statement, for example:

```
Score=82; Qualification=NA

if (Score < 60) {
    Qualification='Failed'
} else {
    if (Score <=80) {
        Qualification='Good'
    } else{
        Qualification='Excellent'}
}
```

At first, variable Score is checked, if **Score** $<$ **60** is true, we asign 'Failed' to variable Qualification, otherwise we check if its smaller or equal to 80, if yes, Qualification will be assigned 'Good', otherwise 'Excellent'.

Meet multiple conditions

The score quantification problem can also be addressed by the vectorized ifelse() statement:

```
Score=c(60, 88, 82, 51, 75, 65)
Quantification=ifelse(Score>=60,
                      ifelse(Score<=80, 'Good','Excellent'), 'Failed')
Quantification

## [1] "Good"      "Excellent" "Excellent" "Failed"    "Good"      "Good"
```

Loops

Loops are used to repeat the same commands again and again with only minor changes in some arguments such as enumarating a vector or doing simulations.

Three types of loops:

- for (elements in object) {do something}
- while (condition) {do something}
- repeat {do something}

Two control keywords:
(i) break: The keyword break breaks at once and exits from the loop.
(ii) next: The keyword next jumps to the start of the next iteration in the loop.

Loops–for

For loops are use when you need to repeat a task for a fixed time, a simple example for loop looks like this:

```
Months=c('January','February','March','April','May','June','July',
         'August','September','October','Nobember','December')
for ( i in 1:length(Months)){
    print(paste(Months[i], ' is the ', i,'th months of the year')) }
```

```
## [1] "January  is the  1 th months of the year"
## [1] "February  is the  2 th months of the year"
## [1] "March  is the  3 th months of the year"
## [1] "April  is the  4 th months of the year"
## [1] "May  is the  5 th months of the year"
## [1] "June  is the  6 th months of the year"
## [1] "July  is the  7 th months of the year"
## [1] "August  is the  8 th months of the year"
## [1] "September  is the  9 th months of the year"
## [1] "October  is the  10 th months of the year"
## [1] "Nobember  is the  11 th months of the year"
## [1] "December  is the  12 th months of the year"
```

Loops—for

Why do we need loops when everything in R is vectorized?

A lot of operations in R can be done in a vectorzed manner, which makes loops unnecessary in some cases, but loops are still useful when:

- The next computation depends on the previous one
- A lot of things need to be done in each round of the loop
- Nested loops

Here's an example of an unnecessary loop:

```
x=1:6
powerx_2=c()
for (i in 1:6){
    powerx_2[i]=2^i
}
#but actually we can just do
2^x

## [1]  2  4  8 16 32 64
```

Loops

Tow ways when enumerating a vector in a loop:

```
names=c('John', 'Wang', 'Linda')
k=1;
for (i in names){
    print(paste('Hello',i, 'student number', k))
    k=k+1
}

## [1] "Hello John student number 1"
## [1] "Hello Wang student number 2"
## [1] "Hello Linda student number 3"

##########################################
names=c('John', 'Wang', 'Linda')
for (i in 1:length(names)){
    print(paste('Hello',names[i], 'student number', i))
}

## [1] "Hello John student number 1"
## [1] "Hello Wang student number 2"
## [1] "Hello Linda student number 3"
```

Loops—repeat

The Repeat loop executes the same code again and again undil some condition it met to brake the loop, it is often used in simulations. For example, we keep flip 3 coins at once, unless they all showed heads:

```
set.seed(100)
repeat{
    coin1=sample(c('head','tail'), 1)
    coin2=sample(c('head','tail'), 1)
    coin3=sample(c('head','tail'), 1)
    print(paste0('coin 1 showed ', coin1,
                 ', coin 2 showd ', coin2,
                 ', and coin 3 showed ',coin3))
    if (all(c(coin1, coin2, coin3)=='head')) break
}

## [1] "coin 1 showed head, coin 2 showd head, and coin 3 showed tail"
## [1] "coin 1 showed head, coin 2 showd head, and coin 3 showed head"
```

Because repeat does not need a condition, so one has to make sure to break the loop when some condition is met.

Loops—while

When using while loop in while(condition){statements} the statements will be repeated unless the condition becomes false, for example:

```
total_apple=100
students=rep(0, 6)
names(students)=c('John', 'Wang', 'Michael', 'Mary', 'Linda', 'Sophia')
while(total_apple>0){
    who=sample(1:6, 1)
    students[who]=students[who]+1
    total_apple=total_apple-1
}

students

##    John    Wang Michael    Mary   Linda  Sophia
##       7      22      20      17      18      16
```

The total number of apples keeps decreacing by one in each round, when it gets to 0, the condition is no longer fulfilled, the while loop breaks.

Exercises:

- Given a integer number a, determine if it is a odd number
- Modify the 'greeting' code so that it says 'Gute nacht' between 10pm to 1am
- Use a while loop to append elements to A=c(0,1) till A has 10 elements, the element appended to A in each round should always equal to the sum of the last two elements of A
- Keep rolling two 6-sided dices and print the result untill both rolled 6 at the same time
- Write a for loop to calculate 1-10+100-1000+...+100000000