For loops, IFELSE and Regex

Jennifer Chicchi

10/01/2019

```
setwd('C:/Users/Canidium User/Desktop/Canidium/Code files')
```

For loops

- It's a good idea to practice making for loops in R.
- This is an example to count the number of even numbers in a vector:

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

```
## [1] 3
```

- Loops are iterative structures that execute a sequence of code n times.
- They have the following components:

  1) A vector of values to 'iterate' over

  2) An index that keeps track of the current iteration

  3) Code to execute

```
for(i in 1:5){
        print(i*10)
}
```

```
## [1] 10
## [1] 20
## [1] 30
## [1] 40
## [1] 50
```

- In the example above, the vector of values is c(1, 2, 3, 4, 5) aka 1:5
- Index is 'i'

- Code to execute on each iteration is 'print(i*10)'
- Typically the vector of values to iterate over will be incremental in units of 1, such as above.
- However, it doesn't have to be. See below where are vector of values is 1, 3, 5, 8.

```
for(i in c(1,3,5,8)){
        print(i*10)
}
```

```
## [1] 10
## [1] 30
## [1] 50
## [1] 80
```

The vector of values need not start at 1.

```
for(i in 3:5){
        print(i*10)
}
```

```
## [1] 30
## [1] 40
## [1] 50
```

- You can also name the index arbitrarily (by convention people use i and j).
- But it can be any variable name, such as 'our_index'

```
for(our_index in 1:5){
        print(our_index*10)
}
```

```
## [1] 10
## [1] 20
## [1] 30
## [1] 40
## [1] 50
```

Let's read in heights file to explore loops using some data on heights of husband-wife pairs.

```
heights <- read.delim('spouseheights.txt')
str(heights)
```

```
## 'data.frame':    96 obs. of  2 variables:
##  $ husband: int  186 180 160 186 163 172 192 170 174 191 ...
##  $ wife   : int  175 168 154 166 162 152 179 163 172 170 ...
```

```
head(heights)
```

```
##   husband wife
## 1     186  175
## 2     180  168
## 3     160  154
## 4     186  166
## 5     163  162
## 6     172  152
```

- Bad use of for loop header (because 'hard-coding' the number of df rows).
- You can use the form 'df$column[index]' to get the i-th value of that column.
- heights is the data frame, what follows $ is the variable name.
- $ tells us return the wife column.
- The index is a location-that is in brackets.

```
for(i in 1:5){
        print(paste("Wife", i, "height =", heights$wife[i]))
}
```

```
## [1] "Wife 1 height = 175"
## [1] "Wife 2 height = 168"
## [1] "Wife 3 height = 154"
## [1] "Wife 4 height = 166"
## [1] "Wife 5 height = 162"
```

Good use of for loop header (because NOT hard-coding the number of df rows).

```
for(i in 1:nrow(heights)){
        print(paste("Wife", i, "height =", heights$wife[i]))
}
```

```
## [1] "Wife 1 height = 175"
## [1] "Wife 2 height = 168"
## [1] "Wife 3 height = 154"
## [1] "Wife 4 height = 166"
## [1] "Wife 5 height = 162"
```

Etc. up to Wife 96 height.

- Use for loop to calculate and print difference of husband-wife height replacing 96 with nrow-the number of rows(heights).
- Tomorrow if we get e.g. 97 rows, we don't have to change the for loop
- The husband's height at the first row, the wife's height at the first row, etc. through to the ith row

```
for(i in 1:nrow(heights)){
        h.height = heights$husband[i] #indexing by row of column
        w.height = heights$wife[i]
        diff = h.height-w.height
        print(paste("The height difference in pair", i, "is", diff, "inches")
)
}
```

```
## [1] "The height difference in pair 1 is 11 inches"
## [1] "The height difference in pair 2 is 12 inches"
## [1] "The height difference in pair 3 is 6 inches"
## [1] "The height difference in pair 4 is 20 inches"
## [1] "The height difference in pair 5 is 1 inches"
```

The column 1, 2 etc. is the specific location here but could do 'husband' or 'wife' instead-slightly more informative

```
for(i in 1:nrow(heights)){
        h.height = heights[i,1] #indexing by specific [row,col] point
        w.height = heights[i,2]
        diff = h.height-w.height
        print(paste("The height difference in pair", i, "is", diff, "inches")
)
}
```

```
## [1] "The height difference in pair 1 is 11 inches"
## [1] "The height difference in pair 2 is 12 inches"
## [1] "The height difference in pair 3 is 6 inches"
## [1] "The height difference in pair 4 is 20 inches"
## [1] "The height difference in pair 5 is 1 inches"
```

Etc. up to height difference in pair 96

- Filling up a storage bin with a for loop.

- Sidenote: could also create an empty storage bin of type character or logical.
- For logical, FALSE is represented by a 0, TRUE by a 1.

```
storage.bin <- numeric(20) # create empty storage bin of length 20 with numer
ic data type
storage.bin

##  [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

for(i in 1:length(storage.bin)){
        storage.bin[i] <- i*10
}
storage.bin

##  [1]  10  20  30  40  50  60  70  80  90 100 110 120 130 140 150 160 170
## [18] 180 190 200
```

Part of the storage bin is empty.

```
storage.bin <- numeric(20)
storage.bin

##  [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

for(i in 1:10){
        storage.bin[i] <- i*10
}
storage.bin

##  [1]  10  20  30  40  50  60  70  80  90 100   0   0   0   0   0   0   0
## [18]   0   0   0
```

Modifying that storage bin.

```
for(i in 1:length(storage.bin)){
        storage.bin[i] <- floor(rnorm(1, mean = 8, sd = 4))
}
storage.bin

##  [1]  5  9  8  4  3 16  3 10  9  3  7  4  4  8 11 16 11 10  7 16
```

Nested for loops:

- You can nest a for loop within another for loop.
- Note: MUST use two separate indices (e.g. i and j).

- Output will be 1 80 90 100 2 80 90 100 3 80 90 100.

```
for(i in 1:3){
        print(i)
        for(j in 8:10){
                print(j*10)
        }
}
```

```
## [1] 1
## [1] 80
## [1] 90
## [1] 100
## [1] 2
## [1] 80
## [1] 90
## [1] 100
## [1] 3
## [1] 80
## [1] 90
## [1] 100
```

IFELSE

- It's a good idea to practice making if/else statements in R.
- Here is a really basic if/else statement:

```
x <- -5
if(x > 0){
        print("Positive number")
} else {
        print("Negative number")
}
```

```
## [1] "Negative number"
```

- If/else statements are simple.
- It follows this format:
    - If a condition is met execute some code.
    - Otherwise execute some other code.

```
x <- 1
if (x > 0){
        print('Hello')
```

```
} else {
        print('Goodbye')
}
```

```
## [1] "Hello"
```

In this example, let's assign grades to our students:

```
grades <- c(75, 80, 85, 90, 95, 100, 88, 92, 78, 72, 65, 60, 40, 20, 50, 74,
84, 0, 100, 88)

grade_bin <- character(20) #Create an empty storage bin for storing letter gr
ades

a_grade <- 74
if (a_grade > 90){ #first rule out the smallest subset of conditions: only 10
values
        print('A')
} else if (a_grade > 80){ #then rule out a bigger subset: 20 values
        print('B')
} else if (a_grade > 70){
        print('C')
} else if (a_grade > 60){
        print('D')
} else {      #if it doesn't meet any of the above conditions
        print('F')
}
```

```
## [1] "C"
```

Using a for loop to assign grades to each of the students based on the grades vector:

```
grades2 <- c(75, 80, 85, 90, 95, 100, 88, 92, 78, 72, 65, 60, 40, 20, 50, 74,
84, 0, 100, 88)
for (i in grades2){
        if (i >= 90){
                print(LETTERS[1])
        } else if (i >= 80){
                print(LETTERS[2])
        } else if (i >= 70){
                print(LETTERS[3])
        } else if (i >= 60){
                print(LETTERS[4])
        } else {
                print(LETTERS[6])
```

```
        }
}
```

```
## [1] "C"
## [1] "B"
## [1] "B"
## [1] "A"
## [1] "A"
## [1] "A"
## [1] "B"
## [1] "A"
## [1] "C"
## [1] "C"
## [1] "D"
## [1] "D"
## [1] "F"
## [1] "F"
## [1] "F"
## [1] "C"
## [1] "B"
## [1] "F"
## [1] "A"
## [1] "B"
```

- Tall couples (both greater than 175)-having both together is going to be a subset.
- Fewer tall couples (both husband and wife) than just tall husband or tall wife.
- if/else useful for huge datasets where you need to classify things.

```
couples.height.bin <- character(nrow(heights))
for(i in 1:nrow(heights)){
        if(heights[i, 'husband'] > 175 & heights[i, 'wife'] > 175){
                couples.height.bin[i] <- 'tall couple'
        } else if (heights[i, 'husband'] > 175){
                couples.height.bin[i] <- 'tall husband'
        } else if (heights[i, 'wife'] > 175){
                couples.height.bin[i] <- 'tall wife'
        } else {
                couples.height.bin[i] <- 'short couple'
        }
}
table(couples.height.bin)
```

```
## couples.height.bin
## short couple  tall couple tall husband
##           48            9           39
```

- If logic within for loops
- Check if number is even using the modulus (%%) operator
- Modulus gives you the remainder-useful for randomizing something

```
10 %% 2
```

```
## [1] 0
```

```
9 %% 2
```

```
## [1] 1
```

```
9 %% 4
```

```
## [1] 1
```

```
9 %% 5
```

```
## [1] 4
```

```
storage.bin <- numeric(10)
for (i in 1:10){
        if(i%%2 == 0){
                storage.bin[i] <- "even"
        } else {
                storage.bin[i] <- "odd"
        }
}
storage.bin
```

```
##  [1] "odd"  "even" "odd"  "even" "odd"  "even" "odd"  "even" "odd"  "even"
```

Check whether the husband or wife is taller:

```
for(i in 1:nrow(heights)){
        diff = heights[i,1] - heights[i,2]
        if(diff > 0){
                print("Husband is taller.")
        } else {
                print("Wife is taller.")
        }
}
```

```
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
```

```
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Wife is taller."
## [1] "Wife is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Wife is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
```

```
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Wife is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Wife is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
## [1] "Husband is taller."
```

- Count how many couples in which male height exceeds female height.
- Declare count in global environment and update it within for loop.
- counter- a way to keep a running total-count up by 1 only when difference is greater than 0.

```
count <- 0
for (i in 1:nrow(heights)){
        diff = heights[i,1] - heights[i,2]
        if (diff > 0){
                count <- count + 1
        }
}

count
```

```
## [1] 91
```

Count how many couples in which female height equals or exceeds male height:

```
tallerWifeCount = 0
for (i in 1:nrow(heights)){
        diff = heights[i,1] - heights[i,2]
        if (diff <= 0){
                tallerWifeCount <- tallerWifeCount + 1
        }
}

count
```

```
## [1] 3
```

REGEX

- Text cleaning finds patterns in strings-e.g. scan for phone numbers or credit card numbers in databases.
- Therefore, avoid PCI violations.
- A set of character matching patterns.
- Can Google these rules as needed.
- get regular expression=grep.
- . (dot): matches any single character, as shown below.

```
strings <- c("^ab", "ab", "abc", "abd", "abe", "ab 12", "ab13")
strings
```

```
## [1] "^ab"   "ab"    "abc"   "abd"   "abe"   "ab 12" "ab13"
```

```
grep("ab.", strings, value = TRUE)
```

```
## [1] "abc"   "abd"   "abe"   "ab 12" "ab13"
```

```
grep(".ab", strings, value = TRUE)

## [1] "^ab"

grep(".ab.", strings, value = TRUE)

## character(0)

grep("ab..", strings, value = TRUE)

## [1] "ab 12" "ab13"

grep("ab...", strings, value = TRUE)

## [1] "ab 12"
```

- [...]: a character list, matches any one of the characters inside the square brackets.
- We can also use - inside the brackets to specify a range of characters.

```
grep("ab[c-e]", strings, value = TRUE)

## [1] "abc" "abd" "abe"

grep("ab[cde]", strings, value = TRUE)

## [1] "abc" "abd" "abe"

grep("ab[cde1]", strings, value = TRUE)

## [1] "abc"  "abd"  "abe"  "ab13"

grep("ab[ce]", strings, value = TRUE)

## [1] "abc" "abe"
```

- [^...]: an inverted character list, similar to [...], but matches any characters
- except those inside the square brackets.

```
grep("ab[^c]", strings, value = TRUE)

## [1] "abd"   "abe"   "ab 12" "ab13"
```

- \: Suppress the special meaning of metacharacters in regular expression, i.e. $ * + . ? [ ] ^ { } | ( ) , similar to its usage in escape sequences.
- Since itself needs to be escaped in R, we need to escape these metacharacters with double backslash like \\$.

```
grep("^ab", strings, value = TRUE)
```

```
## [1] "ab"    "abc"   "abd"   "abe"   "ab 12" "ab13"
```

```
grep("\\^ab", strings, value = TRUE) #using escape sequence
```

```
## [1] "^ab"
```

|: an 'or' operator, matches patterns on either side of the |

```
grep("abc|abd", strings, value = TRUE)
```

```
## [1] "abc" "abd"
```

Character classes

- Character classes allows to specify entire classes of characters, such as numbers, letters, etc.
- There are two flavors of character classes, one uses [: and :] around a predefined name inside square brackets and the other uses and a special character.
- They are sometimes interchangeable.

```
more_strings <- c('123', '123abc', '2019-08-28', '90%', 'hello', 'Hello', 'HI
!!!',
                  'goodBye', 'CAPITAL LETTERS', '$100')
```

digit or digits, 0 1 2 3 4 5 6 7 8 9, equivalent to 0-9

```
grep("[0-9]", more_strings, value = TRUE)
```

```
## [1] "123"         "123abc"      "2019-08-28" "90%"         "$100"
```

```
grep("\\d", more_strings, value = TRUE)
```

```
## [1] "123"         "123abc"      "2019-08-28" "90%"         "$100"
```

```
grep("[[:digit:]]", more_strings, value = TRUE)
```

```
## [1] "123"         "123abc"      "2019-08-28" "90%"         "$100"
```

[:lower:]: lower-case letters, equivalent to [a-z]

```
grep("[a-z]", more_strings, value = TRUE)
```

```
## [1] "123abc"  "hello"   "Hello"   "goodBye"
```

```
grep("[[:lower:]]", more_strings, value = TRUE)
```

```
## [1] "123abc"  "hello"   "Hello"   "goodBye"
```

[:alpha:]: alphabetic characters, equivalent to [[:lower:][:upper:]] or [A-z]

```
grep("[A-Z]", more_strings, value = TRUE)
```

```
## [1] "Hello"           "HI!!!"           "goodBye"         "CAPITAL LETTERS
"
```

```
grep("[[:upper:]]", more_strings, value = TRUE)
```

```
## [1] "Hello"           "HI!!!"           "goodBye"         "CAPITAL LETTERS
"
```

[:alpha:]: alphabetic characters, equivalent to [[:lower:][:upper:]] or [A-z]

```
grep("[[:alpha:]]", more_strings, value = TRUE)
```

```
## [1] "123abc"          "hello"           "Hello"           "HI!!!"
## [5] "goodBye"         "CAPITAL LETTERS"
```

```
grep("[[:lower:][:upper:]]", more_strings, value = TRUE)
```

```
## [1] "123abc"          "hello"           "Hello"           "HI!!!"
## [5] "goodBye"         "CAPITAL LETTERS"
```

```
grep("[A-z]", more_strings, value = TRUE)
```

```
## [1] "123abc"          "hello"           "Hello"           "HI!!!"
## [5] "goodBye"         "CAPITAL LETTERS"
```

[:alnum:]: alphanumeric characters, equivalent to [[:alpha:][:digit:]] or [A-z0-9]

```
grep("[[:alnum:]]", more_strings, value = TRUE)
```

```
##  [1] "123"             "123abc"          "2019-08-28"
##  [4] "90%"             "hello"           "Hello"
##  [7] "HI!!!"           "goodBye"         "CAPITAL LETTERS"
## [10] "$100"
```

```
grep("[[:alpha:][:digit:]]", more_strings, value = TRUE)
```

```
##  [1] "123"             "123abc"          "2019-08-28"
##  [4] "90%"             "hello"           "Hello"
##  [7] "HI!!!"           "goodBye"         "CAPITAL LETTERS"
## [10] "$100"
```

```
grep("[A-z0-9]", more_strings, value = TRUE)
```

```
##  [1] "123"             "123abc"          "2019-08-28"
##  [4] "90%"             "hello"           "Hello"
##  [7] "HI!!!"           "goodBye"         "CAPITAL LETTERS"
## [10] "$100"
```

\w: word characters, equivalent to [[:alnum:]] or [A-z0-9]

```
grep("\\w", more_strings, value = TRUE)
```

```
##  [1] "123"             "123abc"          "2019-08-28"
##  [4] "90%"             "hello"           "Hello"
##  [7] "HI!!!"           "goodBye"         "CAPITAL LETTERS"
## [10] "$100"
```

```
grep("[A-z0-9_]", more_strings, value = TRUE)
```

```
##  [1] "123"             "123abc"          "2019-08-28"
##  [4] "90%"             "hello"           "Hello"
##  [7] "HI!!!"           "goodBye"         "CAPITAL LETTERS"
## [10] "$100"
```

\W: not word, equivalent to [^A-z0-9_]

```
grep("[^A-z0-9_]", more_strings, value = TRUE)
```

```
## [1] "2019-08-28"      "90%"             "HI!!!"           "CAPITAL LETTERS
"
## [5] "$100"
```

```
grep("[[:blank:]]", more_strings, value = TRUE)
```

```
## [1] "CAPITAL LETTERS"
```

- [:space:]: space characters: tab, newline, vertical tab, form feed, carriage return, space.
- You might want to remove in all whitespace e.g. data entry errors w/person typing in zipcode

```
grep("[[:space:]]", more_strings, value = TRUE)
```

```
## [1] "CAPITAL LETTERS"
```

- [:punct:]: punctuation characters, ! " # $ % & â ( ) * + , - . / : ; < = > ? @ [ ] ^ _ ` { | } ~.
- Removing punctuations= cleaning text

```
grep("[[:punct:]]", more_strings, value = TRUE)
```

```
## [1] "2019-08-28" "90%"        "HI!!!"      "$100"
```

gsub(pattern, replacement, looks in string or vector of strings)

```
gsub("[[:punct:]]", 'xxx', more_strings)
```

```
##  [1] "123"             "123abc"          "2019xxx08xxx28"
##  [4] "90xxx"           "hello"           "Hello"
##  [7] "HIxxxxxxxxx"     "goodBye"         "CAPITAL LETTERS"
## [10] "xxx100"
```

function will just remove all punctuation from data

```
gsub("[[:punct:]]", '', more_strings)
```

```
##  [1] "123"             "123abc"          "20190828"
##  [4] "90"              "hello"           "Hello"
##  [7] "HI"              "goodBye"         "CAPITAL LETTERS"
## [10] "100"
```