Introduction to Computing for Economics and Management

Lecture 7: Loops Continued
Previous lecture: looping

The most frequently used looping construct is:

- `for(x in vec) {expression}`

- The for-loop iterates through all elements of the vector `vec`.

- For each element of the vector `vec` there will be one iteration of the loop and `expression` is executed.

- At each iteration, the variable `x` takes the value of the current element of `vec`:
  - First iteration: `x = vec[1]`
  - Second iteration: `x = vec[2]`
  - ...
Previous lecture: print variable when iterating

Let’s print out the value of variable $x$ when iterating through the vector `vec`

```r
> vec <- c(1:5)
> vec
[1] 1 2 3 4 5
> for(x in vec) {print (x)}
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
```
Previous lecture: print variable when iterating

- The for-loops works with other modes beside numeric as well

Example: like before we print the value of the variable when iterating through a vector of strings

```r
> word.vector <- c("a", "text", "consists", "of")
> for(word in word.vector) {print (word)}
[1] "a"
[1] "text"
[1] "consists"
[1] "of"
```
Previous lecture: print variable when iterating

As an alternative we can create a new vector which ranges from 1 until the length of the vector, iterate through this vector and access the original vector via indexing

```r
> vector.indices <- 1:length(word.vector)

> vector.indices
[1] 1 2 3 4

> for(i in vector.indices) {print(word.vector[i])}
[1] "a"
[1] "text"
[1] "consists"
[1] "of"
```
Previous lecture: print variable when iterating

We can write the alternative way in one line

```r
> for(i in 1:length(word.vector)) {print(word.vector[i])}
[1] "a"
[1] "text"
[1] "consists"
[1] "of"
```
Previous lecture: compute length of a vector

Write our own function for computing the length of a vector

```r
## function to compute length of vector vec
vec.length <- function(vec)
{
  # initialize counter
  counter <- 0

  # iterate through vec and increase counter
  for(x in vec) {counter <- counter + 1}

  # return counter
  return(counter)
}
```
Previous lecture: compute Euclidean norm of a vector

```r
## compute Euclidean norm of a vector vec
Euclid.norm <- function(vec)
{
  # initialize norm
  norm <- 0

  # compute sum of squared vector elements
  for(x in vec) {norm <- norm + x^2}

  # sqrt of sum
  norm <- sqrt(norm)

  return(norm)
}
```
Previous lecture: p-norm of a vector

```r
## compute p-norm of a vector vec
p.norm <- function(vec, p=2)
{
  # initialize norm
  norm <- 0

  # compute sum of exponentiated vector elements
  for(x in vec) {norm <- norm + x^p}

  # p radical of sum
  norm <- (norm)^(1/p)

  return(norm)
}
```
Program today

- Loops cont'd
- Read data from file
- New data type: factor

- Nested loops

- Alternative loop constructs
  - While
  - Repeat

- Loop control
  - Break
  - Next
Square elements of a vector

We can change the elements of the input vector and return a new vector, e.g. square the elements of a vector

```r
## square elements of vector vec
square.vec <- function(vec)
{
  # initialize output vector vec.res
  vec.res <- vector()

  # fill vec.res with squared elements of vec
  ????

  return(????)
}
```
Square elements of a vector

We can change the elements of the input vector and return a new vector, e.g. square the elements of a vector

```r
## square elements of vector vec
square.vec <- function(vec)
{
  # initialize output vector vec.res
  vec.res <- vector()

  # fill vec.res with squared elements of vec
  for(x in vec) {vec.res <- c(vec.res, x^2)}

  return(vec.res)
}
```
Square elements of a vector

We test our function

```r
> square.vec(c(1,2,3))
[1] 1 4 9

> square.vec(7:10)
[1] 49 64 81 100

> (7:10)^2
[1] 49 64 81 100
```
Variable access in functions

- Cannot access the variables of functions from outside

```r
> f <- function(x){
  print (paste("x:",x));
  x<-x+1
  z<-x+1;
  print (paste("z:",z));
}

> x <- 10;
> f(x)
[1] ??
[1] ??

> x
[1] ??

> z
[1] ??
```
Function findwords

Now we are ready to program our word list function

```r
## finds locations of each word in word.vec
findwords <- function(word.vec)
{
  # initialize word list
  word.list <- list()

  # iterate through word vector
  for(i in 1:length(word.vec))
  {
    # store current word in variable word
    word <- word.vec[i]
    # add current word to word.list
    word.list[[word]] <- c(word.list[[word]], i)
  }

  return(word.list)
}

> findwords(c("a", "text", "consists", "of", "a"))
```
Function `findwords`

We test our function

```r
> findwords(c("a", "text", "consists", "of", "a"))
$a
[1] 1 5

$text
[1] 2

$consists
[1] 3

$of
[1] 4
```
Read data from file

- The only drawback for now is that we still have to provide a vector of words

- It would be much more convenient if the function would read the text from a file

- We can use the `scan()` function which read data from a file into a vector
Read data from file

- As a first step we create a new text file using RStudio or an alternative text editor like Notepad++
Next, we write our text into the new file
Read data from file

- Finally we save the file
Working directory

Before the actual import, we need to check the current working directory to make sure which path to use when importing the data file

> getwd()
[1] "/home/c7031082/R"

- We change working directory to the path where our data file is located in order to simplify data import
Change working directory in R
Change working directory in RStudio

- In the files tab we select the “…” item and browse to the folder in which we have stored the text file.
Change working directory in RStudio

- In the menu “More”, we select “Set As Working Directory”
Read data from file

We read text data from a file into a vector

```r
> word.vec <- scan("text.txt", "")
Read 15 items

> word.vec
[1] "a"       "text"     "consists" "of"
[5] "a"       "word"     "and"     "another"
[9] "word"    "and"     "so"       "on"
[13] "and"    "so"      "forth"
```

- The second argument is a short form of `what=""` to indicate that we intend to import text data
- Similar like in `read.table()` the separator between items is `white space` by default: one or more spaces, tabs, newlines or carriage returns
### Function findwords

Now, we can use the imported word vector

```r
> findwords(word.vec)
$a
[1] 1 5
$text
[1] 2
$consists
[1] 3
$of
[1] 4
$word
[1] 6 9
...
```
Function findwords

- As a next step, we can enhance our function `findwords` by adding the text file import functionality.

- In this way we don’t need to import the text into a vector beforehand.

- Instead of a word vector, the enhanced function needs a file name as input.
Function findwords

```r
## finds locations of each word in file
findwords <- function(file) {
  # fill word.vec from data in file
  word.vec <- scan(file, "")

  # initialize word list
  word.list <- list()

  # iterate through word vector
  for(i in 1:length(word.vec)) {
    # store current word in variable word
    word <- word.vec[i]
    # add current word to word.list
    word.list[[word]] <- c(word.list[[word]], i)
  }

  return(word.list)
}
```
Function findwords

We test our enhanced function

```r
> findwords("text.txt")
$a
[1] 1 5

$text
[1] 2

$consists
[1] 3

$of
[1] 4

$word
[1] 6 9

...
Sort word list by word frequency

- Now we are ready to obtain the word frequencies from any text.

- The output of our function can be improved.

- So far, we report words in the order how the words appeared in the text.

- It would be more convenient to sort the result alphabetically or by word frequency.
Sort word list by word frequency

- In a previous lecture we have already learned how to sort our word list by word frequency

- First, we determine the word frequency

```r
> word.freq <- sapply(word.list, length)
```

- Next, we compute the order of the word frequency

```r
> word.freq.order <- order(word.freq)
```

- Finally, we use the obtained order of the word frequency for sorting our word list

```r
> word.list[word.freq.order]
```
Sort word list by word frequency

- We can write all three steps in one line
  ```r
  > word.list[order(sapply(word.list, length))]
  ```

- We can revert the sorting order by specifying the argument `decreasing = T` in the `order` function
  ```r
  > word.list[order(sapply(word.list, length), decreasing = T)]
  ```
Sort word list alphabetically

- In a previous lecture we have already learned how to sort our word list alphabetically

- First, we obtain the words from `word.list` using `names()`
  ```r
  > words <- names(word.list)
  ```

- Second, we sort the words alphabetically
  ```r
  > words.sorted <- sort(words)
  ```

- Finally, we sort `word.list` using `words.sorted`
  ```r
  > word.list[words.sorted]
  ```

- We can write all three steps in one line
  ```r
  > word.list[sort(names(word.list))]
  ```
Function findwords

- We enhance our function `findwords` by sorting the word list alphabetically per default.

- We further enhance our function `findwords` by providing an additional sort option `sort.by.freq`.

  - Default value of `sort.by.freq = FALSE` and thus word list is sorted alphabetically by default.

  - In case `sort.by.freq = TRUE` we will sort the word list by word frequency.
In order to implement the sorting feature, we need a control flow construct with the following functionality:

- Check the value of the variable `sort.by.freq`
- In case the condition `sort.by.freq = TRUE` is satisfied, sort by word frequency else sort alphabetically
- A control flow construct which provide this functionality is the so called if-else statement

```plaintext
if (condition) {expression1} else {expression2}
```
- Depending on whether `condition` is true, the result is `expression1` or else `expression2`
if-else

> x <- 2
> y <- if(x == 2) x else x+1
> y
[1] 2

> x <- 3
> y <- if(x == 2) x else x+1
> y
[1] 4

> x <- 3
> y <- if(x == 2){z<-5; x} else {x+1}
> y
[1] 4
> z
### if-else

- In our findwords function we need to check the value of the variable `sort.by.freq`

In case the condition `sort.by.freq = TRUE` is satisfied, we sort by word frequency else we sort alphabetically.

```r
if(sort.by.freq)
{
  # sort by word frequency
  return(word.list[order(sapply(word.list, length), decreasing = T)])
}
else
{
  # sort alphabetically
  return(word.list[sort(names(word.list))])
}
```
Function findwords

```r
## finds locations of each word in file
findwords <- function(file, sort.by.freq = F)
{
  # fill word.vec from data in file
  word.vec <- scan(file, "")
  # initialize word list
  word.list <- list()

  # iterate through word vector
  for(i in 1:length(word.vec))
  {
    # store current word in variable word
    word <- word.vec[i]
    # add current word to word.list
    word.list[[word]] <- c(word.list[[word]], i)
  }
  # sort by word frequency or else sort alphabetically
  if(sort.by.freq)
  {
    return(word.list[order(sapply(word.list, length), decreasing = T)])
  }
  else
  {
    return(word.list[sort(names(word.list))])
  }
}
```
Function findwords

> findwords("text.txt")
Read 15 items
$a
[1] 1 5

$and
[1] 7 10 13

$another
[1] 8

$consists
[1] 3

$forth
[1] 15
...
Function `findwords`

```r
> findwords("text.txt", sort.by.freq=T)
Read 15 items
$and
[1]  7 10 13

$a
[1]  1  5

$word
[1]  6  9

$so
[1] 11 14

$text
[1]  2
...
```
Plotting word frequencies

From the resulting list of word positions returned by `findwords` we can easily calculate the word frequencies using `sapply`

```r
> word.list <- findwords("text.txt", sort.by.freq=T)
Read 15 items

> word.freq <- sapply(word.list, length)

> word.freq

      and       a      word     so     text
     3       2       2       2       1
  consists       of      another    on     forth
     1       1       1       1       1
```
Plotting word frequencies

We create a barplot of the word frequencies

> barplot(word.freq, las=2)
Word frequency in Wikipedia

- Since the for loop allows us to iterate through large texts, let’s apply our function to Wikipedia

- We select the Wikipedia article about the R programming language

- We copy the article in a text editor

- We apply findwords and we create a barplot of the 10 most frequent words
We copy the Wikipedia article about R programming language and save it as `R_wikipedia.txt`.
Word frequency in Wikipedia

> word.list <- findwords("R_wikipedia.txt", sort.by.freq=T)
Read 3395 items

> word.freq <- sapply(word.list, length)

> barplot(word.freq[1:10], las=2)
Word frequency in Wikipedia

The bar chart shows the frequency of different words in Wikipedia. The words are arranged in descending order of frequency, with 'R' being the most frequent and 'in' being the least frequent.
Homework

1. Write a function that iterates through a vector and computes the sum of vector’s elements

2. Write a function that iterates through all columns and rows of a matrix and computes the means of the columns and the rows

3. Create a barplot of the 10 most frequent used words in your favorite Wikipedia article