EEE356 Data Analytics

Laboratory Session 4
Functions, Decision and Repetition Structures

Dr Kasım Zor

20^{th} March, 2023

1. Functions, Conditional Statements, and Some Conventions

Printing some letters: Original

```
giveletter <- function(thenumber){
  if (thenumber == 1){
    theletter <- 'a'
  }
  if (thenumber == 2){
    theletter <- 'b'
  }
  return(theletter)
}
```

```
giveletter(1)
```

```
## [1] "a"
```


1. Functions, Conditional Statements, and Some Conventions

Printing some letters: Original

```
giveletter <- function(thenumber){
  if (thenumber == 1){
    theletter <- 'a'
  }
  if (thenumber == 2){
    theletter <- 'b'
  }
  return(theletter)
}
```

```
giveletter(2)
```

```
## [1] "b"
```


1. Functions, Conditional Statements, and Some Conventions

Printing some letters: Original

```
giveletter <- function(thenumber){
  if (thenumber == 1){
    theletter <- 'a'
  }
  if (thenumber == 2){
    theletter <- 'b'
  }
  return(theletter)
}
```

```
mynumber = 2
giveletter(mynumber)
```

```
## [1] "b"
```

1. Functions, Conditional Statements, and Some Conventions

Printing some letters: Revised

```
giveletter <- function(thenumber){
  if (thenumber == 1){
    theletter <- 'a'}
  else if (thenumber == 2){
    theletter <- 'b'}
  else{
    theletter <- ' '}
  return(theletter)
}
```

```
giveletter(3)
```

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


1. Exercises

1. Execute the followings and observe the results.

```
i <- 5
j <- 6
k <- i + j
print(k)
```

```
## [1] 11
```

```
j <- j + 2
k <- k + j
print(k)
```

```
## [1] 19
```

```
k <- k * k
print(k)
```

```
## [1] 361
```


1. Exercises

2. Write a program that finds and returns the larger one of two given numbers.

1. Exercises

2. Write a program that finds and returns the larger one of two given numbers.

```
givelarger <- function(i, j){
  if (i > j){
    thelarger <- i
  }
  else {
    thelarger <- j
  }
  return(thelarger)
}
```

```
givelarger(3, -4)
```

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

1. Exercises

2. Write a program that finds and returns the larger one of two given numbers.

```
givelarger <- function(i, j){
  if (i > j){
    return(i)
  }
  else {
    return(j)
  }
}
```

```
givelarger(3, -4)
```

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


1. Exercises

3. Write an improved program that finds and returns the larger one of two given numbers. As opposed to the program in the previous exercise, the program should print "the numbers are equal" and return nothing if the inputs are equal. Test your program (after saving and sourcing it) for various inputs.

1. Exercises

3.

```
givelarger <- function(i, j){
  if (i > j){
    return(i)
  }
  else if (j > i){
    return(j)
  }
  else{
    print('The numbers are equal')
  }
}
```

```
givelarger(3, -4)
```

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


1. Exercises

3.

```
givelarger <- function(i, j){
  if (i > j){
    return(i)
  }
  else if (j > i){
    return(j)
  }
  else{
    print('The numbers are equal')
  }
}
```

```
givelarger(-4, 3)
```

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

1. Exercises

3.

```
givelarger <- function(i, j){
  if (i > j){
    return(i)
  }
  else if (j > i){
    return(j)
  }
  else{
    print('The numbers are equal')
  }
}
```

```
givelarger(2, 2)
```

```
## [1] "The numbers are equal"
```


1. Exercises

4. Execute the followings and observe the results.

```
print(pi)
```

```
## [1] 3.141593
```

```
pi <- 3
print(pi)
```

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

```
rm(pi)
print(pi)
```

```
## [1] 3.141593
```


1. Exercises

5. Create a 4x3 matrix in the R workspace as

```
A = matrix(c(1:12), nrow = 4, byrow = 'TRUE')
```

Then, access to different elements of the matrix as follows:

```
A[1, 1:3]
A[1:4, 2]
A[3, 3]
A[11]
A[20]
A[5, 4]
A[1, 1, 1]
```

and explain what happens in each case.

1. Exercises

5. Create a 4x3 matrix in the R workspace as

```
A = matrix(c(1:12), nrow = 4, byrow = 'TRUE')
```

```
A[1, 1:3]
```

```
## [1] 1 2 3
```

```
A[1:4, 2]
```

```
## [1]  2  5  8 11
```

```
A[3, 3]
```

```
## [1] 9
```

1. Exercises

5. Create a 4x3 matrix in the R workspace as

```
A = matrix(c(1:12), nrow = 4, byrow = 'TRUE')
```

```
A[11]
```

```
## [1] 9
```

```
A[20]
```

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


1. Exercises

5. Create a 4x3 matrix in the R workspace as

```
A = matrix(c(1:12), nrow = 4, byrow = 'TRUE')
```

```
A[5, 4]
```

```
## Error in A[5, 4]: subscript out of bounds
```

```
A[1, 1, 1]
```

```
## Error in A[1, 1, 1]: incorrect number of dimensions
```


2. Loops

Consider the calculation of

$$\sum_{i=1}^{n} |v[i]|$$

for a given vector $v \in \mathbb{R}^n$. The vector has n elements. The most trivial algorithm to compute the 1-norm can be described as follows:

- Initialize a sum value as zero.
- Add the absolute value of the first element to the sum value.
- Add the absolute value of the second element (if n $>$ 2) to the sum value.
- ...
- Add the absolute value of the last element to the sum value.
- Return the sum value.

2. Calculation of 1-Norm Using For

```
onenorm_for <- function(v){
  sumvalue <- 0
  for (i in 1:length(v)) {
    sumvalue <- sumvalue + abs(v[i])
  }
  return(sumvalue)
}
```

```
v <- c(1:20)
onenorm_for(v)
```

```
## [1] 210
```

2. Calculation of 1-Norm Using While

```
onenorm_while <- function(v){
  sumvalue <- 0
  i <- 1
  while (i <= length(v)) {
    sumvalue <- sumvalue + abs(v[i])
    i <- i + 1
  }
  return(sumvalue)
}
```

```
onenorm_while(v)
```

```
## [1] 210
```


2. Loops

Lets assume that we would like to find the location of the first zero element of a vector $v \in \mathbb{R}^n$ by using for and while statements.

2. Finding the First Zero Using For

```
findzero_for <- function(v){
  for (i in 1:length(v)) {
    if (v[i] == 0){
      return(i)
    }
  }
}
```

```
v <- c(100:1,0,-1:-20)
findzero_for(v)
```

```
## [1] 101
```


2. Finding the First Zero Using While

```
findzero_while <- function(v){
  i <- 1
  while (v[i] != 0 && i <= length(v)) {
    i <- i + 1
  }
  if (i <= length(v)){
      return(i)
  }
  else {
    print('The vector does not contain any zero element.')
  }
}
```

```
findzero_while(v)
```

```
## [1] 101
```

2. Finding the First Zero Using While

```
findzero_while <- function(v){
  i <- 1
  while (v[i] != 0 && i <= length(v)) {
    i <- i + 1
  }
  if (i <= length(v)){
      return(i)
  }
  else {
    print('The vector does not contain any zero element.')
  }
}
```

```
q <- c(1:1200)
findzero_while(q)
```

```
## [1] "The vector does not contain any zero element."
```


2. Matrix-Vector Multiplication

Consider the multiplication of a matrix $A \in \mathbb{R}^{m \times n}$ with a vector $x \in \mathbb{R}^n$. If $y = Ax$, we have

$$y[i] = \sum_{j=1}^{n} A[i,j]x[j]$$

for $i = 1, 2, \ldots, m$.

2. Matrix-Vector Multiplication

```
matvectmult <- function(A, x){
  m <- nrow(A)
  n <- ncol(A)
  y <- matrix(0, nrow = m)
  for (i in 1:m) {
    sumvalue <- 0
    for (j in 1:n) {
      sumvalue <- sumvalue + A[i, j]*x[j]
    }
    y[i] <- sumvalue
  }
  return(y)
}
```


2. Matrix-Vector Multiplication

```
B <- matrix(c(1,2,3,4,5,6,7,8,9), nrow = 3, ncol = 3)
B
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
```

```
z <- matrix(c(1,2,3), nrow = 1, ncol = 3)
z
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
```

```
matvectmult(B, z)
```

```
##      [,1]
## [1,]   30
## [2,]   36
## [3,]   42
```

3. Recursion: Fibonacci Numbers

Fibonacci numbers starting from 1 can be written as

$$1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \ldots,$$

where each number (except for the first two numbers) is the sum of two previous numbers. This recurrence relation can be written as

$$f(i) = f(i-1) + f(i-2), \quad i > 2$$

with conditions $f(1) = 1$ and $f(2) = 2$.

3. Recursive Calculation of ith Fibonacci Number

```
fiborecursive <- function(i){
  if (i <= 2){
    value <- 1
  }
  else {
    value1 <- fiborecursive(i-1)
    value2 <- fiborecursive(i-2)
    value  <- value1 + value2
  }
  return(value)
}
```

```
fiborecursive(12)
```

```
## [1] 144
```


3. Recursion: Factorial

The recurrence relation

$$i! = i \times (i-1)!$$

results in

$$0! = 1$$

for the termination condition.

The general formula can be given as

$$i! = i \times (i-1) \times (i-2) \times \ldots \times 1$$

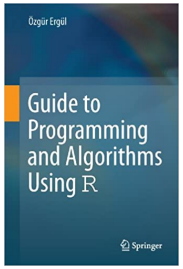
3. Recursive Calculation of Factorial

```
factorialrecursive <- function(i){
  if (i == 0){
    value <- 1
  }
  else {
    value <- i*factorialrecursive(i-1)
  }
  return(value)
}
```

```
factorialrecursive(5)
```

```
## [1] 120
```

References



Aforementioned contents are adapted from the book:

- Guide to Programming and Algorithms Using R

which is written by

- Prof. Özgür Ergül
- Middle East Technical University
- Department of Electrical and Electronics Engineering