Instituto Politécnico Nacional

Escuela Superior de Computo

Programación para la ciencia de datos.

Cristal Karina Galindo Durán

Practica 3:

PROGRAMACIÓN DE FUNCIONES

Vianey Maravilla Pérez

3AM1

Unidad temática a la que corresponde la práctica

I. Programación orientada al manejo de datos en Lenguaje R

Objetivo

Realizar scripts en Lenguaje R que permita incluir funciones para resolver problemas de manera modular.

Introducción:

Una función es un subprograma que realiza una tarea especifica que puede o no recibir valores (parámetros). La utilidad de realizar funciones dentro de los programas es reutilizar código; asimismo, darle legibilidad al código desarrollado.

En esta practica se incluyen un conjunto de ejercicios que le permiten al docente poner en práctica la creación de funciones en Lenguaje R.

1. Escribe una función que dado un valor y un vector devuelva cuántas veces se encuentra el valor en el vector.

Procedimiento:

Para lograr hacer este script, tuve que ocupar mis conocimientos de funciones básicas, ya que implementar 4 funciones para que cada función pudiera hacer una instrucción cada una, aparte de la principal que es la que contiene la llamada a las funciones.

Consideraciones:

Para este ejercicio se puede implementar de dos maneras:

- vector<-c(1, 2, 3, 4,3) > vector %in% 3 [1] FALSE FALSE TRUE FALSE TRUE
- numeros=sample(1:10, size=cant, replace= TRUE, prob= NULL)

Programa:

```
"Ejercicio 1: Escribe una función que dado un valor y un vector
devuelva cuántas veces se encuentra el valor en el
vector

Creado por: Maravilla Pérez Vianey 3AM1"

vector<-function()
{
vector_a<- generador()
cat (vector_a)
buscador<- enumero()
imprimir_num<- bnumero(vector_a, buscador)
}

generador<-function()
{
cant =as.integer(readline("¿Cuantos numeros deseeas generar dentro de este vector?\n"))
numeros=sample(1:10, size=cant, replace= TRUE, prob= NULL)
vector_a<-c(numeros)
}

enumero<- function()
{
buscador= as.integer(readline("Escoge el numero a econtrar 1-10\n"))

}

bnumero<-function(vector_a, buscador)
{
imprimir_num<- vector_a[vector_a==buscador]
cat("\n\n El numero", buscador ," Se repitio",imprimir_num, "veces")

}
```

RESULTADOS:

```
Console  Terminal  Jobs
R 4.1.1 · ~/
>
> vector<-function()
+ {
+ vector_a<- generador()
+ cat (vector_a)
+ buscador<- enumero()
+ imprimir_num<- bnumero(vector_a, buscador)
+ }
>
> generador<-function()
+ {
+ cant =as.integer(readline("¿Cuantos numeros deseeas generar dentro de este vector?\n"))
+ numeros=sample(1:10, size=cant, replace= TRUE, prob= NULL)

+ vector_a<-c(numeros)
+ }
>
> enumero<- function()
+ {
+ buscador= as.integer(readline("Escoge el numero a econtrar 1-10\n"))
+
+ }
>
> bnumero<-function(vector_a, buscador)
+ {
+ imprimir_num<- vector_a[vector_a==buscador]
+ cat("\n\n El numero", buscador ," Se repitio",imprimir_num, "veces")
+
+ }
> vector()
¿Cuantos numeros deseeas generar dentro de este vector?
10
7 8 9 10 6 7 6 5 10 1
Escoge el numero a econtrar 1-10
1


 El numero 1  Se repitio 1 veces
```

2. Una función recibe como parámetro una matriz. La matriz representa los tiempos empleados por un ciclista en varias etapas. Cada fila representa una etapa. La primera columna de la matriz almacena el número de horas, la segunda columna el número de minutos y la tercera columna el número de segundos que tardó en completar la etapa. Por ejemplo, si se recibe:

2	30	50
1	55	20

El ciclista ha completado dos etapas. En la primera etapa ha tardado 2 horas, 30 minutos y 50 segundos. La función debe devolver una lista con el número total de horas, minutos y segundos empleados por el ciclista en cubrir el total de etapas. Para los datos de ejemplo se devolvería 4 horas, 26 minutos y 10 segundos.

Procedimiento:

Para este script se realizaron funciones con matrices que ya vimos la practica pasada, sin embargo, con un grado de complejidad un poco más alto ya que se hizo un traslado de los datos, y eso complico el procedimiento

Consideraciones:

Se tomaron en cuenta los conocimientos de programación de funciones, así como las funciones básicas que ya vimos, además se tomó mucho en cuenta al usuario en la consola, además que implementamos operaciones básicas.

Programa:

```
"Una funcion recibe como parametro una matriz. La matriz representa los tiempos empleados por un ciclista en varias etapas.
Cada fila representa una etapa. La primera columna de la matriz almacena el número de horas, la segunda columna el número de
minutos y la tercera columna el número de segundos que tardó en completar la etapa. Porejemplo, si se recibe
2 30 50
1 55 20
El ciclista ha completado dos etapas. En la primera etapa ha tardado 2 horas, 30 minutos y 50 segundos.
La función debe devolver una lista con el número total de horas, minutos y segundos empleados por el ciclista en cubrir el total de etapas
Para los datos de ejemplo se devolvería 4 horas, 26 minutos y 10 segundos.
Creado por: Vianey Maravilla Pérez Vianey 3AM1"

Principal<-function()
{
h1<-pedirh1()
m1<-pedirm1()
s1<-pedirs1()
etapa<-crear1(h1, m1, s1)

h2<-pedirh2()
m2<-pedirh2()
s2<-pedirs2()
etapados<-crear2(h2, m2, s2)

total<-generador()(etapa, etapados)
print("\n\n",total)
suma<-sumastotales(h1, m1, s1, h2, m2, s2, etapa, etapados)
}

pedirh1<-function()
{
h1= as.integer(readline("Introduce las horas que tuvo el ciclista en la etapa uno"))
}

pedirm1<-function()
{
m1= as.integer(readline("Introduce los minutos que tuvo el ciclista en la etapa uno del 0 a 59 min"))
}
```

```
pedirs1<-function()
{
  s1= as.integer(readline("Introduce los segundos que tuvo el ciclista en la etapa uno del 0 a 59 seg"))
}

crear1<-function(h1, m1, s1)
{
etapa<-c(h1, m1, s1)

}

pedirh2<-function()
{
  h2= as.integer(readline("Introduce las horas que tuvo el ciclista en la etapa dos"))
}

pedirm2<-function()
{
  m2= as.integer(readline("Introduce los minutos que tuvo el ciclista en la etapa dos del 0 a 59 min"))
}

pedirs2<-function()
{
  s2= as.integer(readline("Introduce los segundos que tuvo el ciclista en la etapa dos del 0 a 59 seg"))
}

crear2<-function(h2, m2, s2)
{
  etapados<-c(h2, m2, s2)

}

generador<-function(etapa, etapados)
{
total<- matrix(c(etapa, etapados),  nrow = 2, byrow = T)
}

sumastotales<-function(h1, m1, s1, h2, m2, s2, etapa, etapados)
{

hTotales<- h1+h2
mTotales<- m1+m2
if(mTotales>60)
{
hTotales<-hTotales+1
mTotales<-(m1+m2)-60
}
stotales<-s1+s2
if(stotales>60)
{
mTotales<-mTotales+1
stotales<-(s1+s2)-60
}

sumas<-c(hTotales, mTotales, stotales)
print("\n\n EL tiempo total de ambas etapas fue",hTotales, "horas", mTotales, "minutos", sTotales, "segundos")

}
```

3. Una matriz es simétrica si es cuadrada y es igual a su transpuesta. Escribe una función que devuelva un valor lógico indicando si una matriz es simétrica.

Procedimiento:

Para este ejercicio lo que hice fue implementar los conocimientos de las funciones y de las funciones ya establecidas para poder generar la transpuesta de la matriz

Consideraciones:

Se necesita saber las funciones de las matrices pero también se necesita saber cómo hacer los números aleatorios para poder generar una matriz, es similar a C

Programa:

```
"Una matriz es simétrica si es cuadrada y es igual a
su transpuesta. Escribe una función que devuelva
un valor lógico indicando si una matriz es simétrica.
Creado por: Vianey Maravilla Pérez 3AM1"

Principal1() <- function(ma,maT){
  for(i in 1:5){
    for( j in 1:5){
      print(isTRUE(ma[i][j] == ma_T[i][j]))
    }
  }

}


ma<- matrix(1:100, nrow = 5, ncol = 5)
print("ORIGINAL")
ma
maT <- t(matriz)
print("TRANSPUESTA")
maT
```

RESULTADO

```
[1] "ORIGINAL"
> ma
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    6   11   16   21
[2,]    2    7   12   17   22
[3,]    3    8   13   18   23
[4,]    4    9   14   19   24
[5,]    5   10   15   20   25
> maT <- t(matriz)
> print("TRANSPUESTA")
[1] "TRANSPUESTA"
> maT
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
[4,]   10   11   12
>
```

CONCLUSIONES:

Al terminar esta practica se pudo comprobar y reafirmar los conocimientos con funciones dentro de una programación de problemas específicos distintos de los vistos en clase, sin embargo, se entendió que se debe ser autónomo e investigar cosas más complejas para poder llevar la programación a otro nivel.