

19/Noviembre/2018

COMO CREAR Y USAR FUNCIONES EN R



Nombre: Santiago Bravo Molina

De control: 18480441

¿CÓMO SE DEFINE UNA FUNCIÓN EN R?

Cuando se realiza un código que involucra muchas líneas de programación y donde además hay partes que son similares, es preferible escribir el código utilizando funciones. Una o varias funciones pueden ser almacenadas en un archivo para luego ser empleadas dentro de un código principal la función sea requerida. Por ejemplo, en un código hay que resolver diversos sistemas lineales. Copiar y pegar el pedazo de código que resuelve el sistema hace el código extremadamente extenso y difícil de leer. Sin embargo, si introducimos una función que resuelva sistemas lineales, cada vez que necesitemos resolver el sistema lo que tenemos que hacer es usar la función específica ya programada y a lo sumo utilizaremos una sola línea de código.

Para programar en R una función que calcule el volumen de un cilindro dado su altura y su radio,

$$V=\pi r^2 h$$

El volumen V es la variable de salida o output, mientras que el radio r y la altura h son parámetros de entrada (input). Para definir la función en R podemos emplear la siguiente estructura:

```
volumen<-function(r,h){  
volumen<-pi*r^2*h  
return(volumen)  
}
```

La función la podemos colocar en un archivo (script) o dentro del mismo archivo principal, recuerden que la ventaja es que este mismo pedazo de código va ser utilizado varias veces dentro del archivo principal sin necesidad de reescribir el código. Solo es necesario invocar la función,

```
V<-volumen(0.2,0.8)  
print('Volumen          para          r=0.2          y          h=1')  
print(V)
```

De esta manera, cada vez que necesitemos calcular el volumen para nuevos valores de h y r solo tenemos que llamar la función y colocar los parámetros de entrada en el orden correcto. En este ejemplo, primero se coloca el radio y luego la altura.

¿CÓMO CREAR Y USAR FUNCIONES EN R?

Las funciones en R son tratadas como cualquier otro objeto. Para crearlas utilizamos el comando `function()`, el cual crea objetos de tipo `function`, de la siguiente manera:

```
f <- function(<argumentos>)  
{  
  ## Código de la función (function body)  
}
```

Luego, para llamar a la función simplemente escribimos el nombre de esta:

```
f <- function()  
{  
  cat("Hola Mundo")  
}  
f()  
Hola Mundo  
class(f)  
"function"
```

Las funciones poseen 3 partes:

- El cuerpo (body)
- Los argumentos (formals)
- El ambiente (environment)

```
f <- function(x, y) { x + y }
```

```
body(f)
```

```
{  
  x + y
```

```
}
```

```
formals(f)
```

```
$x
```

```
$y
```

```
environment(f)
```

```
<environment: R_GlobalEnv>
```

Funciones en R

R tiene dos modos de trabajo básicos (Interactivo y Batch) sin contar programas de manejo, como Rstudio; el modo interactivo es el que conocemos.

Modo de batch

R tiene dos modos de trabajo básicos (Interactivo y Batch) sin contar programas de manejo como Rstudio; el modo interactivo es el que conocemos cuando iniciamos una sesión de R en windows, este modo es el más "amigable" para los usuarios

nuevos, pero para objetivos de programación es altamente disfuncional y lento.

El modo Batch por el contrario, se encarga de automatizar sesiones en R, correr scripts y ser flexible a nuestras necesidades, sin necesidad de abrir R y seguir POR PASOS y manualmente

una serie de comandos para obtener un producto final. Lo que se debe hacer entonces, es poner el código de nuestro análisis o función dentro de un archivo que la mayoría de veces termina en .R (recomendable), aunque puede ser un .txt, entre otros.

Por ejemplo, creamos un archivo de texto llamado "yayirobe.R" que contenga lo siguiente:

```
-----  
pdf("prueba_hist.pdf") # Doy un nombre al archivo que contendrá el "output" en .pdf  
hist(rnorm(200)) # Genero 200 números y hago un histograma con ellos  
dev.off() # Cierro el archivo que contiene el histograma  
-----
```

Todo lo que esta después de # (numeral) en cada linea de comandos del archivo son comentarios

y estos serán ignorados por el interprete de R, estos comentarios sirven para recordarnos que es lo que estamos haciendo con cada linea de comandos. (solo se ejecutara en R lo que esta

antes del # en cada linea.

Con el modo Batch, es posible ejecutar cientos o miles de ordenes y lineas en R de modo

automático

y rápido, en este caso solo ejecutamos tres líneas de comandos para generar un histograma. Específicamente:

1. pdf("prueba_hist.pdf") --->llamamos la función pdf para decirle a R que queremos guardar el gráfico en un archivo llamado "prueba_hist.pdf"

2. hist(rnorm(200)) --->genero 200 números al azar que tienen distribución normal, con rnorm(random normal) que van de 0 a 1, y hago un histograma con estos números utilizando la función hist

3. dev.off() --->cierro la ventana o "device" en el que se escribirá el histograma al archivo que nombramos anteriormente (prueba_hist.pdf), y el archivo se escribe en el directorio en el cual estemos trabajando.

Finalmente se puede ejecutar este archivo (yayirobe.R) para obtener el pdf con el histograma, de dos formas. En windows, abriendo R y dándole click en las pestañas superiores y escogiendo el archivo.

En linux, como me parece mejor, no hay necesidad de abrir R y se puede ejecutar el archivo directamente desde la consola (konsole) del sistema con el comando:

\$ R CMD BATCH yayirobe.R

Introducción a funciones en r

Una función es un grupo de instrucciones que toma un "input" o datos de entrada, usa estos datos

para computar otros valores y retorna un resultado/producto.

Para empezar con un pequeño ejemplo, definiremos dos funciones con las cuales se puedan calcular

el porcentaje de purinas y pirimidinas en una secuencia de ADN.

Llamaremos al archivo que contiene las funciones "puripiri.R" :

calcular el porcentaje de purinas y pirimidinas en una secuencia de ADN

```
Purinas<-function(x) {  
  Purinas <- 0 # asignar 0 a Purinas  
  for (n in x) {  
    if (n == "A") Purinas <- Purinas + 1 # contar las purinas  
    if (n == "G") Purinas <- Purinas + 1 # contar las purinas  
  }  
  return((Purinas/(length(x)))*100)  
}
```

```
Pirimidinas<-function(x) {  
  Pirimidinas <- 0 # asignar el valor de 100 a Pirimidinas  
  {  
    Pirimidinas <- 100-Purinas(x)  
  }  
  return(Pirimidinas)  
}
```

```
#> Pirimidinas(c("A","T","T","G","G","G"))
```

```
#[1] 33.33333
```

```
#> Purinas(c("A","T","T","G","G","G"))
```

```
#[1] 66.66667
```

```
#> Purinas(c("A"))
```

```
#[1] 100
```

```
#> Pirimidinas(c("A"))
```

#[1] 0

Todas las líneas que se encuentran después de un # son comentarios que se agregan al código de la función. En este caso el primer comentario menciona lo que hacen las funciones que se van a escribir.

Primero se define la primer función y se le da el nombre que se desee y que se aplicara al objeto x (x), para nuestro caso

el nombre es "purinas" y continuación se empieza a escribir el cuerpo de la función y se escribe después de haber abierto un corchete ({}):

Purinas<-function(x) {

Le asignamos un valor de 0 (para empezar) al porcentaje de purinas en la secuencia, y a partir de este valor empezaremos a hacer el conteo de las purinas para cada base en la secuencia:

Purinas <- 0 # asignar 0 a Purinas

Ahora le decimos que para cada elemento n del objeto x, en este caso la secuencia de ADN, se le aplicara

el resto de el código de la función, en pocas palabras, abrimos un loop:

for (n in x) {

A continuación le decimos lo que debe hacer con cada elemento n del objeto x, cada vez que lo evalué.

Para nuestro caso seria: si (if) el objeto(n) que encuentra en la secuencia(x) es "A" o "G", entonces le sume

1 a Purinas, que antes habíamos asignado un valor de 0; Que pase al siguiente elemento(n) de la secuencia(x)

y que vuelva a hacer lo mismo:

if (n == "A") Purinas <- Purinas + 1 # contar las purinas

if (n == "G") Purinas <- Purinas + 1 # contar las purinas

Finalmente le decimos que cierre la parte de operaciones de la función con el corchete (}) y que lo ue la función nos

debe arrojar (return) es el valor del ((número de las bases que sean Purinas, sobre la longitud (length) de la secuencia(x), es decir, el length(x) es el mismo número de elementos(n) de x), multiplicado x 100).

En pocas palabras, le decimos que nos arroje como resultado de la función el porcentaje de bases purinicas que se encuentran en la secuencia de ADN:

```
}  
  return((Purinas/(length(x)))*100)  
}
```

De este modo, cerramos el cuerpo y terminamos de escribir, nuestra primer función. Y empezamos a escribir la segunda función, a la que llamaremos "Pirimidinas":

Pirimidinas<-function(x) {

Al igual que como lo hicimos con la primer función, asignamos una variable llamada Pirimidinas con el valor inicial de 0:

Pirimidinas <- 0 # asignar el valor de 100 a Pirimidinas

Lo siguiente es algo muy importante en R, y es el hecho de que se puede llamar una función dentro de otra función, en nuestro caso llamaremos la función "Purinas" (previamente creada) y a 100 le restaremos el valor de el resultado de la función "Purinas" de una secuencia de ADN (x), puesto que el resto de las bases de las secuencias que no son Purinas, deben explicitamente ser Pirimidinas y como es un porcentaje, el porcentaje de Pirimidinas será 100 menos el porcentaje de Purinas que ya calculamos previamente:

```
{  
  Pirimidinas <- 100-Purinas(x)  
}
```


Finalmente lo que hacemos es decirle a R que nos arroje el resultado de la resta anterior, que es el resultado del porcentaje de Pirimidinas en la secuencia de ADN:

```
return(Pirimidinas)  
}
```

Para finalizar lo que se ponen, son ejemplos de casos en los que se prueba o se utiliza la función, de tal forma que nos aseguremos de que las dos funciones esta escritas y definidas correctamente, estos ejemplos, se escriben en forma de comentarios precedidos por el símbolo de numeral #:

```
#> Pirimidinas(c("A","T","T","G","G","G"))  
#[1] 33.33333  
#> Purinas(c("A","T","T","G","G","G"))  
#[1] 66.66667  
#> Purinas(c("A"))  
#[1] 100  
#> Pirimidinas(c("A"))  
#[1] 0
```

Variable scope

Una variable/objeto que se crea dentro de una función, es llamada una variable local, puesto que es temporal y solo se utiliza dentro de la función, mientras se efectúan los cálculos u operaciones, y una vez obtenido el resultado esta variable es eliminada.

En nuestra función de ejemplo las variables "Purinas" y "Pirimidinas" a las que les asignamos 0 inicialmente, al igual que la variable n, son variables locales.

De tal modo que cuando llamemos a "n" por fuera de la función, en la consola de R, nos dira que el objeto 'n' no existe y que no lo encuentra.

y si llamo a "Pirimidinas" en la consola de R, el me dirá que Pirimidinas es una función que llame con ese nombre

, pero no me lo reconoce como una variable por fuera de la función:

```
-----  
> Pirimidinas(c("A","T","T","G","G","G"))  
[1] 33.33333  
> n  
Error: object 'n' not found  
> Pirimidinas  
function(x) {  
  Pirimidinas <- 0 # asignar el valor de 100 a Pirimidinas  
  {  
    Pirimidinas <- 100-Purinas(x)  
  }  
  return(Pirimidinas)  
}  
> care_perro <- 08071988  
-----
```

De manera que se pueden definir variables locales dentro de funciones, que tengan los mismos nombres de variables

globales por fuera de la función o inclusive con el mismo nombre de la función, y R no se confundirá y mantendrá ambas variables

como separadas.

Y finalmente la variable "care_perro" es una variable global que creamos por fuera de una función y que contiene el número

08071988 .

Argumentos por defecto

Consideremos la función:

```
> firulallo <- function(x,y=5,z=F) { ... }
```

Esta función la llamamos "firulallo" y "no tiene cuerpo", puesto que dejamos abierto o vacía la definición de la función *per se* con los corchetes y los tres puntos, ({ ... }).

Mientras tanto, lo que si definimos son argumentos que se ejecutaran por defecto, siempre y cuando el usuario no cambie el argumento o lo re-defina, es decir, para y definimos un valor de 5 y para z, que es una variable lógica la definimos como FALSE o F; de tal forma que si ejecutamos la función sin cambiar los argumentos por defecto, R utilizará los que definimos al escribir la función.

Para un objeto x, con valor a 100, utilizando los argumentos por defecto:

```
>firulallo(100)
```

Para un objeto x, con valor de 100, cambiando los argumentos por defecto:

```
>firulallo(100, y=60, Z=TRUE)
```

Para un objeto x, con valor de 126, cambiando solo un argumento por defecto (el otro argumento, el que no cambiemos, se ejecutara por defecto):

```
>firulallo(126, y=45)
```

Bibliografía

<http://mecabot-ula.org/tutoriales/como-se-define-una-funcion-en-r/>

<https://www.r-bloggers.com/lang/spanish/2512>

https://cran.r-project.org/doc/contrib/rdebuts_es.pdf