

Funciones

Luciano Selzer

28 June, 2018

¿Por qué hacer funciones?

Si nuestros datos fuesen estáticos no haría falta volver a correr nuestro análisis nunca. Pero en general no suele ser así:

- A veces agregamos datos.
- Encontramos errores.
- O tenemos que repetir nuestro análisis porque cada cierto período de tiempo tenemos otro set de datos.

Por ejemplo, **gapminder** es actualizado regularmente.

Las funciones nos permiten reunir operaciones en una unidad. Nos da:

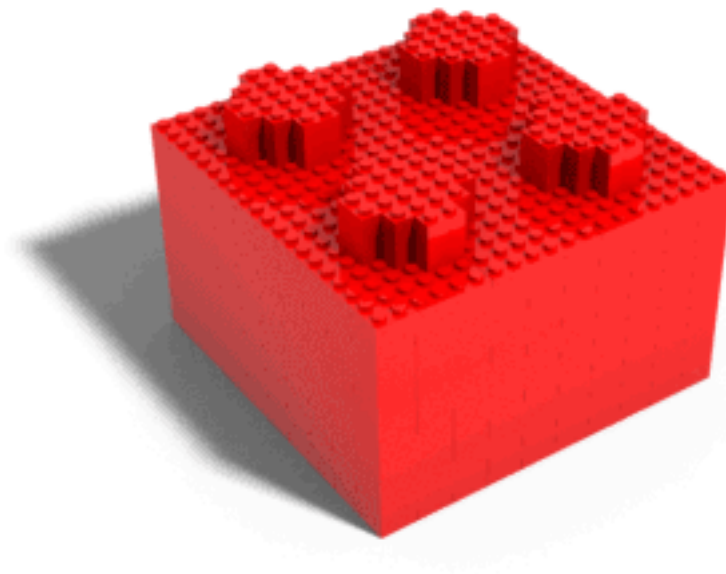
- Un nombre que podemos recordar y llamar.
- Nos evita tener que recordar las operaciones individuales.
- Un conjunto de argumentos de entrada y un valor de salida.
- Conexiones más ricas con el resto del ambiente de programación.

Son el bloque básico de construcción.

Linking to ImageMagick 6.7.7.10

Enabled features: cairo, fontconfig, freetype, fftw, lcms, pango, rsvg, x11

Disabled features: ghostscript, webp



sheepfilms.co.uk

Y si ya has creado una función puedes considerarte programador.

¿Cómo definimos una función en R?

Creemos una nueva carpeta llamada `functions`
Y dentro un archivo `funciones-leccion.R`

```
mi_suma <- function(a, b) {  
  suma <- a + b  
  return(suma)  
}
```

Definamos una función para convertir grados Kelvin en Fahrenheit

```
kelvin_a_fahr <- function(temp) {  
  fahr <- (temp - 273.15) * (9/5) + 32  
  return(fahr)  
}
```

Partes de una función:

- Nombre
- Argumentos
- Cuerpo

En R no es necesario explicitar el `return`.

Automáticamente devuelve el último comando ejecutado. Solo puede devolver **un solo** objeto.

Usemos nuestra función.

```
# El punto de congelación del agua  
kelvin_a_fahr(273.15)
```

```
[1] 32
```

```
# El punto de hervor  
kelvin_a_fahr(373.15)
```

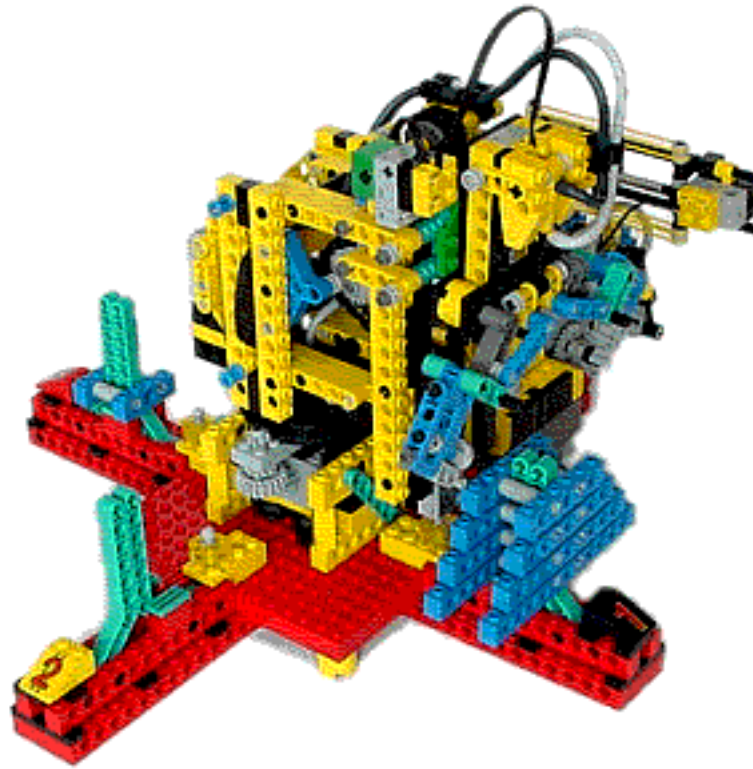
```
[1] 212
```

Ejercicio 1

Crean una función que convierta los grados Celsius a Kelvin

Combinando funciones

El verdadero poder de las funciones surge cuando son combinadas.



Definamos dos funciones:

```
kelvin_a_fahr <- function(temp) {  
  fahr <- (temp - 273.15) * (9/5) + 32  
  return(fahr)  
}  
  
celsius_a_kelvin <- function(temp){  
  kelvin <- temp + 273.15  
  return(kelvin)  
}
```

Ejercicio 2

Crean una función que convierta los grados Celsius en Fahrenheit usando las dos funciones anteriores.

Vamos a crear una función para calcular el producto bruto interno:

```
# Toma el set de datos y multiplica la columna  
# población por PBI per capita  
calcPBI <- function(dat) {  
  pbi <- dat$pop * dat$gdpPercap  
  pbi  
}
```

Usemos nuestra función

```
calcPBI(head(gapminder))
```

```
[1] 6567086330 7585448670 8758855797 9648014150 9678553274 11697659231
```

Nos es muy informativa la salida.

Vamos a añadir argumentos para poder seleccionar el país y el año:

```
# Toma el set de datos y multiplica la columna  
# población por PBI per capita  
calcPBI <- function(dat, year=NULL, country=NULL) {  
  if(!is.null(year)) {  
    dat <- dat[dat$year %in% year, ]  
  }  
  if (!is.null(country)) {  
    dat <- dat[dat$country %in% country,]  
  }  
  gdp <- dat$pop * dat$gdpPercap  
  
  new <- cbind(dat, gdp = gdp)  
  return(new)  
}
```

Ahora podemos ver los datos de PBI para un año específico:

```
head(calcPBI(gapminder, year = 2007))
```

| | country | year | pop | continent | lifeExp | gdpPercap | gdp |
|----|-------------|------|----------|-----------|---------|------------|--------------|
| 12 | Afghanistan | 2007 | 31889923 | Asia | 43.828 | 974.5803 | 31079291949 |
| 24 | Albania | 2007 | 3600523 | Europe | 76.423 | 5937.0295 | 21376411360 |
| 36 | Algeria | 2007 | 33333216 | Africa | 72.301 | 6223.3675 | 207444851958 |
| 48 | Angola | 2007 | 12420476 | Africa | 42.731 | 4797.2313 | 59583895818 |
| 60 | Argentina | 2007 | 40301927 | Americas | 75.320 | 12779.3796 | 515033625357 |
| 72 | Australia | 2007 | 20434176 | Oceania | 81.235 | 34435.3674 | 703658358894 |

Un país específico:

```
head(calcPBI(gapminder, country = "Argentina"))
```

| | country | year | pop | continent | lifeExp | gdpPercap | gdp |
|----|-----------|------|----------|-----------|---------|-----------|--------------|
| 49 | Argentina | 1952 | 17876956 | Americas | 62.485 | 5911.315 | 105676319105 |
| 50 | Argentina | 1957 | 19610538 | Americas | 64.399 | 6856.856 | 134466639306 |
| 51 | Argentina | 1962 | 21283783 | Americas | 65.142 | 7133.166 | 151820757737 |
| 52 | Argentina | 1967 | 22934225 | Americas | 65.634 | 8052.953 | 184688236498 |
| 53 | Argentina | 1972 | 24779799 | Americas | 67.065 | 9443.039 | 233996596624 |
| 54 | Argentina | 1977 | 26983828 | Americas | 68.481 | 10079.027 | 271970723960 |

O un país y un año específico:

```
head(calcPBI(gapminder, country = "Argentina", year = 2007))
```

| | country | year | pop | continent | lifeExp | gdpPercap | gdp |
|----|-----------|------|----------|-----------|---------|-----------|--------------|
| 60 | Argentina | 2007 | 40301927 | Americas | 75.32 | 12779.38 | 515033625357 |

Veamos que hemos hecho:

```
calcPBI <- function(dat,  
                    year = NULL,  
                    country = NULL)
```

Nombramos la función como calcPBI.

Y tiene tres argumentos:

- dat que ya lo teníamos
 - year y country cuyos valores por defecto son NULL
-

```
if(!is.null(year)) {  
  dat <- dat[dat$year %in% year, ]  
}  
if (!is.null(country)) {  
  dat <- dat[dat$country %in% country,]  
}
```

Comprobamos si los argumentos son nulos, en caso que no lo sean solo seleccionamos los años o países que están en esos argumentos.

Tip: paso por valor

En R los argumentos de las funciones pasan como valor. R hace copias de los argumentos que son los que operamos dentro de la función. Cuando modificamos gapminder dentro de nuestra función modificamos una copia que solo existe dentro de la función.

Es mucho más seguro escribir código de esta forma porque asegura que los objetos fuera de la función permanecen inalterados.

Tip: ámbito de la función

Otro concepto importante es el ámbito (*scope*) de la función. Todas las variables creadas en el cuerpo de la función solo existen dentro de su ámbito. Incluso si tenemos objetos con el mismo nombre en nuestro espacio de trabajo, estos no van a entrar en conflicto ni van a ser modificados por las variables creadas en nuestra función.

```
gdp <- dat$pop * dat$gdpPercap  
new <- cbind(dat, gdp=gdp)  
return(new)  
}
```

Finalmente, calculamos el PBI y lo unimos a nuestros datos.

Ejercicio 3

Prueba nuestra función `PBI` calculando el PBI de Nueva Zelanda en 1987 ¿Cómo difiere del PBI de Nueva Zelanda en 1952?

Ejercicio 4

La función `paste()` puede ser usada para combinar texto.

```
mejores_practicas <- c("Escribe", "programas", "para", "personas",  
                      "no", "para", "computadoras")  
paste(mejores_practicas, collapse = " ")
```

```
[1] "Escribe programas para personas no para computadoras"
```

Ejercicio 4

Escribe una función llamada `vallar`, con dos argumentos `texto` y `envoltura`, e imprime el `texto` envuelto con la `envoltura`.

```
vallar(texto = mejores_practicas, envoltura = "***")
```

Nota: la función `paste` tiene un argumento `sep`, que especifica el separador de texto, con valor por defecto `" "`. `paste0` no tiene espacios por defecto.

Tip: Pruebas y Documentación

Es muy importante probar y documentar las funciones. La documentación te ayuda a vos y a otros, a entender el propósito de la función, como usarla, como funciona. Y es importante asegurarse de que la función hace lo que uno cree que hace.

Tip: Pruebas y Documentación

Cuando recién comenzas, tu forma de trabajar seguramente se parecerá a algo así:

1. Escribe la función
2. Comenta partes de la función para documentar su comportamiento.
3. Carga la fuente del código
4. Experimenta con la consola para asegurarte que se comporta como esperas.
5. Arregla los posibles errores.
6. Enjuaga y repite.

Tip: Pruebas y Documentación

La documentación formal para funciones, escrita en un archivo separado `.Rd`, se convierte en la documentación que ves en las ayudas de las funciones. El paquete `roxygen2` permite a los programadores de escribir la documentación junto con el código de la función y luego procesarlo en los archivos `.Rd` apropiados. Querrás cambiar a este método más formal cuando empieces a escribir proyectos de R más complejos.

Para la automatización de las pruebas se puede usar el paquete `testthat`