



Factores, dataframes y funciones

Gutiérrez Cruz Abel Isaías

Factors

Factors son usados para representar datos

- No ordenados: Mujer y hombre
- Ordenados: Ellos tienen un orden pero no son numéricos
 - Ejemplo: profesores asistentes, profesores asociados y profesores de tiempo completo

Se puede pensar en los factores como un vector de enteros donde cada entero tienen una etiqueta

- Factores son tratados de forma especial por funciones de modelado como "lm()" y "glm()"
- Son muy descriptivos

Crear un factor



```
x <- factor(c("yes", "yes", "no", "yes", "no"))
```

```
# ver cuantas veces aparece un factor
```

```
table(x)
```

Ver el orden de los factores



```
unclass(<factor>)
```

Ordenar los factores



```
x <- factor(c("yes", "yes", "no", "yes", "no"), levels = c("yes", "no"))
```


Dataframe

Data frames son usados para guardar datos tabulados

- Son representados como un especial tipo de lista. Cada elemento de la lista debe de tener la misma longitud
- Pueden contener elementos de diferente tipo en cada una de sus columna (principal diferencia del dataframe y matriz)

Crear un Dataframe:

- Leer un archivos de datos
- Crearlo en R:



```
x <- data.frame(foo = 1:4, bar = c(T, T, F, F))
```

Obtener el número de filas o de columnas

útil también para matrices



```
# número de filas  
nrow(<dataframe>)
```

```
# número de columnas  
ncol(<dataframe>)
```

Obtener el nombre de las columnas



```
names(<dataframe>)
```

Obtener el nombre de las filas



```
row.names(<dataframe>)
```

Creación de un dataframe vacío

```
data.frame(nombre = character(),  
            temperatura = numeric(),  
            dia = numeric())
```

Acceder a filas y columnas

- Se puede hacer uso del signo "\$" seguido del nombre de la columna
- Se puede colocar el nombre o número de la fila o columna entre corchetes "[]"

Usar "\$":

```
<dataframe>$<nombre_de_la_columna>
```


Acceder a la fila 1:



```
<dataframe>[1, ]
```

Acceder a la columna 1:



```
<dataframe>[, 1]
```

Acceder a un fila con un nombre determinado



```
<dataframe>["<nombre de la fila>", ]
```

Acceder a una columna con un nombre determinado



```
<dataframe>[, "<nombre_de_la_columna>"]
```

Agregar filas y columnas



```
x <- data.frame(foo = 1:4, bar = c(T, T, F, F))
```

```
# Agregar una fila
```

```
rbind(x, list(1, TRUE))
```

```
# Agregar una columna
```

```
metros <- c(1, 2, 3, 4)
```

```
cbind(x, metros)
```

Eliminar determinadas columnas



```
x <- data.frame(foo = 1:4, bar = c(T, T, F, F),  
                saludo = c("hola", "hola", "adios", "adios"))  
  
# Eliminar la columna 2 y 3  
x <- x[, -c(1, 3)]  
# Seleccionar solo las columnas foo y saludo  
x <- x[, c("foo", "saludo")]
```

Funciones

Modularidad

Su principio principal es el "Divide y vencerás"

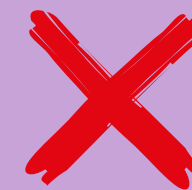
Se trata de un subprograma que se encarga de hacer una tarea en específico. Esta tarea es repetitiva, por lo que cada que el algoritmo principal tenga que llevar a cabo ese procedimiento solo llamará al subprograma.

El subprograma recibe los datos del programa principal y puede devolver o no algún resultado.

Ventajas

- Si el algoritmo que estamos diseñando es largo, esta técnica lo vuelve más simple
- Cada modulo se elabora de forma independiente
- Encontrar bugs es más fácil

Modularidad



Funciones

```
● ● ●  
  
f <- function(<arguments>){  
  # do something  
}
```

Obtener los parámetros de una función

```
● ● ●  
  
# parametros formales  
formals(<funcion>)  
# todos los parametros  
args(<funcion>)
```

Funciones regresando elementos



```
suma <- function(num1, num2){  
  resultado <- num1 + num2  
  resultado  
}
```



```
suma <- function(num1, num2){  
  resultado <- num1 + num2  
  return(resultado)  
}
```


Funciones sin regresar elementos



```
imprimirResultado <- function(num1, num2){  
  resultado <- num1 + num2  
  print(resultado)  
}
```

Parametros



```
multiplicar <- function(num1, num2 = 2){  
  resultado <- num1 + num2  
  print(resultado)  
}
```

Scope

globalNum <- 12

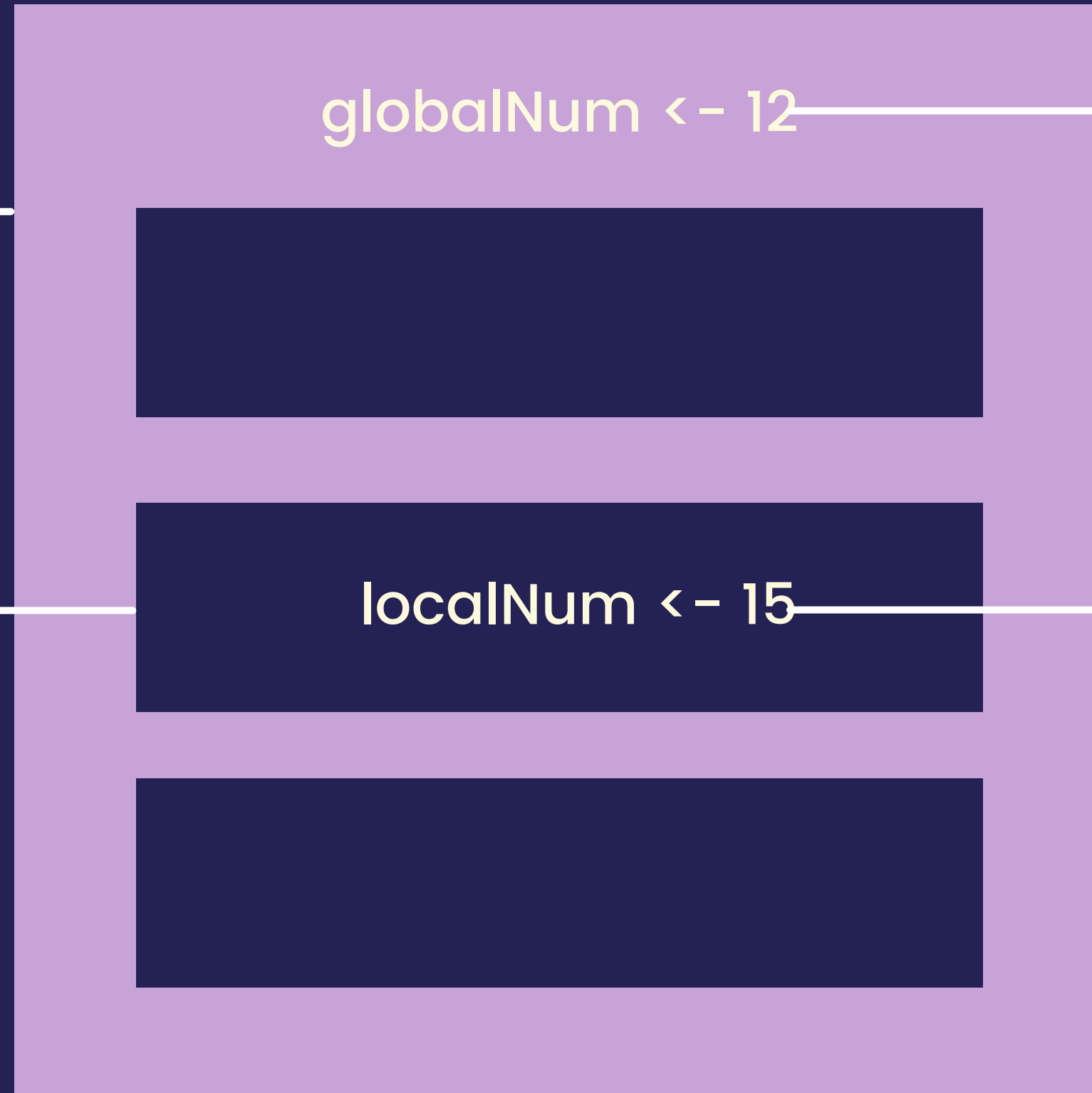
Global variable

Global scope

localNum <- 15

Local variable

Local scope



Scope

- Cuando el programa o la función termina todas sus variables son olvidadas
- Un local scope es creada siempre que una función es llamada

Principales principios

- El código en el global scope no puede usar ninguna variable local
- Un local scope puede acceder a las variables globales
- El código en el local scope de una función no puede usar variable de ningun otro local scope

Funciones anonimas



```
function(<parametros>){a + b}
```

Ejemplo



```
Reduce(function(a, b) a + b, 1:10)
```