PROGRAMACIÓN EN

IF-ELSE

```
if ( condición1 ) {
    ## hacer algo
} else if ( condición2 ) {
    ## hacer otra cosa
} else {
    ## hacer otra cosa diferente
}
```

FOR

```
for (i in 1:10) {
    print(i)
}
```

WHILE

```
count <- 0
while(count < 10) {
    print(count)
    count <- count + 1
}
```

REPEAT

```
x <- 1
repeat {
    print(x)
    x = x+1
    if (x == 6){
    break
    }
}
```

NEXT Y BREAK

`next()`: ignora una iteración

`break`: sale del bucle inmediatamente

FUNCIONES

son objetos de primera clase en R

- se pueden pasar como argumentos a otras funciones
- se pueden anidar
- aceptan parámetros
- devuelven resultados

FUNCIONES

simple:

```
funcion <- function() { #definición
    cat("Hola, mundo!\n")
}
> funcion() #llamada
```

con parámetro `num` y valor devuelto `letras`:

```
funcion <- function(num) { #definición
hola <- "Hola, mundo!\n"
for(i in 1:num) {
    cat(hola)
}
letras <- nchar(hello) * num
letras
}
> funcion(4) #llamada
```

en R, el valor devuelto por una función siempre es la última expresión evaluada

FUNCIONES

con parámetro por defecto:

```
funcion <- function(num = 1) { #definición
hola <- "Hola, mundo!\n"
for(i in 1:num) {
cat(hola)
}
letras <- nchar(hello) * num
letras
}
> funcion() #llamada
```

num es un argumento formal porque está incluido en la definición de la función

los nombres de los argumentos se pueden indicar explícitamente:

```
> funcion(num = 2) # llamada
```

COINCIDENCIA DE ARGUMENTOS

puede ser posicional, por nombre o mixto

```
> str(rnorm)
function (n, mean = 0, sd = 1)
```

posicional

```
> mydata <- rnorm(100, 2, 1)
```

por nombre:

```
> mydata <- rnorm(sd = 1, n=100, mean=2)
```

mixto:

```
> mydata <- rnorm(mean = 2, sd = 1, 100)
```

la coincidencia también puede ser parcial

EL ARGUMENTO . . .

indica un número variable de argumentos que normalmente se pasan a otras funciones

se usa cuando se extiende una función y no se quiere copiar toda la lista de argumentos de la original

```
myplot <- function(x, y, type = "l", ...) {
    plot(x, y, type = type, ...) ## se pasa '...' a la función 'plot'
}
```

es necesario cuando el número de argumentos que se pasa a una función no puede conocerse previamente

```
> args(paste)
> args(cat)
```

los argumentos que vengan después de `...` deben ser nombrados total y explícitamente

EJERCICIO

Escribir una funcion `funcion1` y otra función `funcion2` tales que si x es un vector `(x1,x2...xn)` entonces `funcion(x)` devuelva el vector `(x1, x2^2...xn^n)` y `funcion2(x)` devuelva el vector `(x1, x2^2/2...xn^n/n)`

FUNCIONES BUCLE

iteran en forma compacta

- `lapply()`: itera una lista y evalúa una función en cada elemento
- `sapply()`: igual que la anterior pero trata de simplificar el resultado
- `apply()`: evalúa una función sobre los márgenes de un array
- `tapply()`: evalúa una función en subconjuntos de un vector

LAPPLY()

- itera sobre cada elemento de una lista
- aplica una función (a especificar) a cada elemento de la lista
- devuelve una lista

```
> x <- list(a = 1:5, b = rnorm(10))
> lapply(x, mean)
```

al pasar una función como argumento de una función, no es necesario incluir los paréntesis ( )

SAPPLY()

funciona igual que `lapply()` sólo que:

- si el resultado es una lista donde cada elemento es de longitud 1, devuelve un vector
- si el resultado es una lista donde cada elemento es un vector de la misma longitud (>1), devuelve una matriz
- si no puede adivinar qué está pasando, devuelve una lista

```
> x <- list(a = 1:4, b = rnorm(10), c = rnorm(20, 1), d = rnorm(100, 5))
> lapply(x, mean)
> sapply(x, mean)
```

TAPPLY()

aplica una función sobre subconjuntos de un vector

```
> ## datos simulados
> x <- c(rnorm(10), runif(10), rnorm(10, 1))
> ## definición de grupos con variable factor
> f <- gl(3, 10) ## 3 niveles de 10 repeticiones cada uno
> tapply(x, f, mean)
```

APPLY()

evalúa una función sobre los márgenes de un array

se suele usar para aplicar una función sobre las filas o columnas de una matriz

```
> x <- matrix(rnorm(200), 20, 10)
> apply(x, 2, mean) ## obtener la media de cada columna
> apply(x, 1, sum) ## obtener la suma de cada fila
```

atajos: `rowSums()`, `rowMeans()`, `colSums()`, `colMeans()`

Acceso al repositorio con la presentación

Acceso a la presentación

Fco. Javier Parra Rodríguez,Jefe de Servicio de Estadísticas Económicas y Socio-demográficas