# Práctica 0
# Funciones auxiliares

## Índice



## Para entregar

- Carpeta "auxiliares" con los problemas de la sección 3 resueltos.

## 1. Observaciones para todas las prácticas

### 1.1. Introducción

El entorno de trabajo elegido para realizar las prácticas de la asignatura es R. Los principales motivos que justifican esta elección son:

- Es software libre, cómodo e intuitivo.
- Debido a la gran cantidad de recursos ya desarrollados que suministra, permite programar fácilmente otras funciones.
- Se utiliza en otras asignaturas, por lo que el alumnado está familiarizado con su uso.

### 1.2. Método de trabajo

- Debéis leer con atención los enunciados.
- En el directorio *programas*, que contiene un subdirectorio para cada una de las prácticas, están las funciones que van a ser programadas. Si el código está completo, hay que leerlo con detenimiento. Si no está completo, **se debe completar**.

- Deberéis siempre trabajar en el directorio "programas": Para comprobar el directorio en el que estáis, escribid *getwd*() en la línea de comandos y para cambiar de directorio *setwd*(" ").

  Ejemplo:

```
> getwd()
[1] "/home/.../criptografia/programas/auxiliares"
> setwd("..")
> getwd()
[1] "/home/.../criptografia/programas"
```

- Cada vez que realicéis una entrega, deberá ser de **toda la carpeta** correspondiente. Por ejemplo, esta primera semana deberéis entregar toda la carpeta "auxiliares", aunque sólo hayáis tenido que completar la sección de problemas.

- Sólo se debe realizar una entrega después de comprobar que los programas funcionan correctamente. Para ello, en los enunciados de las prácticas hay propuestos ejemplos y ejercicios con sus respectivas soluciones.

## 1.3. Sentencias y estructuras de R que necesitaremos

A modo de ayuda complementaria, esta documentación pretende presentar estructuras o sentencias de R que serán necesarias para la implementación de cada algoritmo.

La palabra clave para crear una función es *function*. En el ejemplo siguiente creamos la función *bin2dec*(), que admite como parámetro un vector de bits que representa un entero $n$ en forma binaria, con el bit más significativo a la izquierda, y devuelve como salida el entero $n$ expresado en base decimal.

```
source("auxiliares/evalpol.R")

bin2dec <- function(nbin)
  # Entrada: nbin (vector de bits que representa un entero n en
             forma binaria)
  # Salida : valor del entero n en base decimal
{
  # Código
  n=evalpol(nbin,2)
  return(n)
}
```

El carácter `#` sirve para declarar comentarios. R ignorará los caracteres de una línea siguientes a la aparición de `#`.

Este programa hace uso de la función *evalpol*(), previamente programada en el fichero "evalpol.R", y que describiremos más adelante. Por ello, debemos cargar este fichero con la orden *source*(" "), especificando el camino completo. Como siempre trabajaremos en la carpeta "programas" y el fichero que debemos cargar ("evalpol.R") se encuentra en la subcarpeta "auxiliares", la orden será:

```
source("auxiliares/evalpol.R")
```

De la misma forma, para utilizar la función *bin2dec*(), deberemos cargar previamente el fichero donde esté escrita:

```
> source("auxiliares/bin2dec.R")
> bin2dec(c(1, 0, 0, 1))
[1] 9
```

Para crear nuestros programas, necesitaremos hacer uso de operadores matemáticos como éstos:

| Operación | Símbolo |
|---|---|
| Adición | + |
| Substracción | - |
| Multiplicación | * |
| División | / |
| Potenciación | ^ |
| Módulo | %% |
| División de enteros | %/% |

Una sentencia muy utilizada será la sentencia $if$, que tiene esta sintaxis en R:

```
if(condición)
  {
    Flujo de instrucciones 1
  }else
  {
    Flujo de instrucciones 2
  }
```

La segunda parte es opcional. Puede ser

```
if (condición) {Flujo de instrucciones}
```

La condición puede necesitar operadores relacionales como estos:

| Símbolo matemático | Operador de R |
|---|---|
| $=$ | `==` |
| $\neq$ | `!=` |
| $\geq$ | `>=` |
| $>$ | `>` |
| $\leq$ | `<=` |
| $<$ | `<` |

O también, operadores lógicos como estos:

| Operador lógico | Operador de R |
|---|---|
| `AND lógico` | $x$`&`$y$ |
| `OR lógico` | $x$`\|`$y$ |
| `NOT lógico` | `!`$x$ |
| `OR EXCLUSIVO lógico` | `xor`$(x, y)$ |

Otra sentencia que utilizaremos repetidamente será la sentencia *while* que permite construir bucles. Su sintaxis en R es:

```
while(condición)
  {
    Grupo de instrucciones
  }
```

# 2. Funciones auxiliares

A lo largo de este curso necesitaremos utilizar algunas funciones, que están en la carpeta "programas/auxiliares".

A continuación hacemos una descripción de cada una de ellas.

Es conveniente que os familiaricéis con su denominación y funcionamiento, ya que serán utilizadas en las siguientes prácticas. Para ello, deberéis realizar los ejercicios propuestos y comparar las soluciones.

- *evalpol*(): Evalúa un polinomio en un punto. Admite como entrada un vector cuyas componentes son los coeficientes del polinomio, con el coeficiente de grado más alto a la izquierda, y el punto en que va a ser evaluado.

```
evalpol <- function(pol, a)
  # Evalua el polinomio pol en a
  # Entrada: pol (vector de coeficientes del polinomio)
  #          pol=pol[1]*x^g+pol[2]*x^{g-1}+ ....+pol[g]*x+pol[g+1]
  # Salida:  valor del polinomio en a,
  #          val=pol[1]*a^g+pol[2]*a^{g-1}+ ....+pol[g]*a+pol[g+1]
```

EJEMPLO. Si $p(x) = 7x^3 + 14x^2 + 11x$, calculamos $p(26)$:

```
>  p <- c(7, 14, 11, 0)
> evalpol(p, 26)
[1] 132782
```

EJERCICIO. Si $p(x) = 2x^4 + 25$, calcular $p(101)$.
Solución: 208120827.

- *cambiobase*(). Dado un número entero no negativo expresado en base 10, calcula su representación en otra base.

```
cambiobase <- function(x,b)
  # Entrada: x (entero no negativo)
  #          b (entero mayor o igual que 2)
  # Salida:  vector de enteros que representan los
  #          dígitos de x en la base b
```

EJERCICIOS.

1. Calcular la representación en base 5 de 1001. Comprobar el resultado.
   Solución: 1 3 0 0 1. Comprobación: $5^4 + 3 \cdot 5^3 + 1 = 1001$.
2. Calcular la representación binaria (en base 2) de 13. Comprobar el resultado.
   Solución: 1 1 0 1. Comprobación: $2^3 + 2^2 + 1 = 13$.
3. Calcular la representación en base 16 de 161. Comprobar el resultado.
   Solución: 10 1. Comprobación: $10 \cdot 16 + 1 = 161$.

- *dec*2*bin*(): Dado un entero no negativo $n$ expresado en base 10 y un entero positivo $k$, calcula la representación binaria (en base 2) de $n$. La salida es un vector de $k$ bits, con el bit más significativo a la izquierda. Debe ser $k > \log_2 n$.

- *bin*2*dec*(): Dado un vector de bits que representa un entero $n$ en forma binaria, con el bit más significativo a la izquierda, calcula el valor de $n$ en base decimal.

  **Observación.** Las conversiones entre formato decimal y hexadecimal, pueden efectuarse con las funciones *as.hexmode*() y *as.integer*() de R. Observad el doble uso de *as.hexmode*() en el ejemplo siguiente.

  EJEMPLO.

```
> x <- as.hexmode(161)
> x
[1] "a1"
> as.integer(x)
[1] 161
> y <- as.hexmode("2b")
> as.integer(y)
[1] 43
```

  EJERCICIOS.

  1. Calcular la representación binaria (12 bits) y la hexadecimal de 1245. Solución: 0 1 0 0 1 1 0 1 1 1 0 1; $''4dd''$.
  2. La representación binaria de $n$ es 1 1 1 1 1. Calcular $n$ en base decimal. Solución: 31.
  3. La representación hexadecimal de $n$ es $''fea''$. Calcular $n$ en base decimal. Solución: 4074.

- *men*2*num*(). Dado un alfabeto, un mensaje escrito en dicho alfabeto y un entero positivo $k$, calcula el equivalente numérico del mensaje, cuando éste ha sido partido en $k$-gramas (si el número de caracteres del mensaje no es múltiplo de $k$, se añade la última letra del alfabeto tantas veces como sea necesario).

  EJEMPLO. Para calcular el equivalente numérico del mensaje "LA PRUEBA 1 ES", partiéndolo en trigramas y sabiendo que ha sido escrito en el alfabeto

  $$A, B, C, D, E, F, G, H, I, J, K, L, M, N, O,$$
  $$P, Q, R, S, T, U, V, W, X, Y, Z, 0, 1, 2, \text{“ ”}$$

  utilizaremos ($LETTERS$ contiene las 26 letras del alfabeto inglés en mayúsculas):

```
> alfabeto <- c(LETTERS, 0, 1, 2, " ")
> men2num(alfabeto, "LA PRUEBA 1 ES", 3)
[1]  9929 14030  3630 26939  4169
```

EJERCICIO. Calcular el equivalente numérico de "hola", utilizando bloques de una letra y el alfabeto inglés de 26 letras minúsculas (*letters*).
Solución: 7 14 11 0.

- *num*2*men*(): Es la función inversa de *men*2*num*(). Sea $v$ un vector de números enteros procedente de la representación numérica de un mensaje escrito en un cierto alfabeto y dividido en $k$-gramas.

  Dado el alfabeto, el vector $v$ y el entero $k$, devuelve como salida el mensaje.

  EJEMPLO. Para obtener el mensaje cuyo equivalente numérico es

  $$9929\,14030\,3630\,26939\,4169,$$

  sabiendo que ha sido calculado partiéndolo en trigramas y utilizando el alfabeto

  $$A,\ B,\ \ldots,\ Z,\ 0,\ 1,\ 2,\ \text{“ ”}$$

  utilizaremos:

```
> alfabeto <- c(LETTERS, 0, 1, 2, " ")
> num2men(alfabeto, c(9929, 14030,  3630, 26939,  4169), 3)
[1] "LA PRUEBA 1 ES "
```

  EJERCICIO. Obtener el mensaje cuyo equivalente numérico es 7 14 11 0, sabiendo que ha sido calculado partiéndolo en bloques de una letra y utilizando el alfabeto inglés de 26 letras minúsculas (*letters*).
  Solución: "hola".

- *men*2*bit*(): Dado un alfabeto, un mensaje escrito en dicho alfabeto y un entero positivo $k$, calcula el vector con el equivalente binario de los $k$-gramas en que queda partido el mensaje (si el número de caracteres del mensaje no es múltiplo de $k$, se añade la última letra del alfabeto tantas veces como sea necesario).

  EJERCICIO. Calcular el equivalente binario de "ADIOS", utilizando bloques de dos letras y el alfabeto inglés de 26 letras ($LETTERS$)
  Solución: 0 0 0 0 0 0 0 0 1 1 0 0 1 1 0 1 1 1 1 0 0 1 1 1 1 0 1 1 0 1.

- *bit*2*men*(): Es la función inversa de *men*2*bit*(). Dado un alfabeto, un vector *menbit* de bits y un entero positivo $k$, devuelve como salida un mensaje tal que al dividirlo en $k$-gramas y transformar cada $k$-grama en bits, se obtiene el vector *menbit*.

EJERCICIO. Transformar el vector

$$0\,0\,0\,0\,0\,0\,0\,0\,1\,1\,0\,0\,1\,1\,0\,1\,1\,1\,1\,0\,0\,1\,1\,1\,1\,0\,1\,1\,0\,1$$

en mensaje, sabiendo que ha sido calculado partiéndolo en bloques de dos letras y utilizando el alfabeto $LETTERS$.
Solución: "ADIOSZ"

# 3. Problemas

Los siguientes problemas hay que resolverlos escribiendo los comandos necesarios en el fichero "problemasauxiliares.R", después del enunciado correspondiente. A modo de ejemplo, el primer problema está resuelto.

1. Si $p(x) = 1 + 5x^2 + x^5$, calcular $p(2)$.
2. Si $p(x) = 5x^4 + 2x^3 + x^2$, calcular $p(3)$.
3. Calcular la representación en base 8 de 2001.
4. Calcular la representación binaria (con 16 bits) y hexadecimal de 2001.
5. La representación binaria de $n$ es $1\,1\,0\,1\,1$. Calcular su representación decimal y hexadecimal.
6. La representación hexadecimal de $n$ es "e1". Calcular su representación decimal y binaria (10 bits).
7. Utilizando el alfabeto inglés de 26 letras y el espacio:

   $$A,\ B,\ \ldots,\ Z,\ \text{“ ”}$$

   y suponiendo que los mensajes se parten en bigramas:

   *a*) Calcular el equivalente numérico del mensaje

   "EJERCICIOS DE FUNCIONES"

   *b*) Calcular el equivalente binario del mensaje

   "SON FACILES"

   *c*) Obtener el mensaje cuyo equivalente numérico es

   $$121\,74\,121\,530\,404\,297\,720\,389\,542\,230\,377$$

   *d*) Obtener el mensaje cuyo equivalente binario es

   $$0\,0\,1\,1\,0\,0\,0\,0\,0\,1\,1\,0\,1\,0\,1\,1\,1\,1\,1\,0\,0\,0\,0\,0\,0\,1\,1\,0\,1\,1\,0\,0\,0\,0\,0\,0\,0\,1\,1\,0\,1\,1\,0\,0\,0\,1\,0\,1\,1\,1\,1\,1$$