Notas importantes

- Todos los ejercicicios resueltos serán subidos a los repositorios de github. La forma en la que el estudiante debe presentar este laboratorio es:
  - Debes crear una carpeta local llamada `Practica2`.
  - Dentro de la carpeta `Practica2`, se deben crear las carpetas `Ejercicio1, Ejercicio2, Ejercicio3, Ejercicio4,...` donde se deben alojar las soluciones con la extensión `Rmd` para los comentarios que son parte de las preguntas y `.R` del lenguaje R para los otros. Cualquier otra extensión, incluyendo letras minúsculas, será motivo de una rebaja de puntos en el ejercicio.
- Los archivos de respuesta deben llevar un comentario inicial con tu nombre y código. Por ejemplo.

```
> # Nombre : Cesar Lara Avila 20122345J
> # Respuesta1: El codigo  muestra el uso del SVD en el calculo
> #de la inversa de matrices
>
> library(MASS)
>
> a <- matrix(c(1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0,
+     0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1), 9, 4)
>
> a.svd <- svd(a)
> a.svd$d
```

- El laboratorio calificado es individual, cualquier copia de código puede implicar la descalificación de este laboratorio.

Preguntas

1. (a) Identifica los primeros 20 elementos contenidos en el paquete `methods` que se carga automáticamente al iniciar R. ¿Cuántos ítems hay en total?

   (b) Determina el entorno que posee cada una de las siguientes funciones:
   - `read.table`
   - `data`
   - `matrix`
   - `jpeg`

   (c) Usa `ls` y `test` de igualdad de cadena de caracteres para confirmar que la función `smoothScatter` es parte del paquete `graphics`.

2. En cada una de las siguientes líneas de código, identifica qué estilo de coincidencia de argumentos se está utilizando: exacto, parcial, posicional o mixto. Si es mixto, identifica qué argumentos se especifican en cada estilo.

```
> array(8:1,dim=c(2,2,2))
> rep(1:2,3)
> seq(from=10,to=8,length=5)
> sort(decreasing=T,x=c(2,1,1,2,0.3,3,1.3))
> which(matrix(c(T,F,T,T),2,2))
> which(matrix(c(T,F,T,T),2,2),a=T)
```

3. (a) Crea los siguientes dos vectores

```
> vec1 <- c(2,1,1,3,2,1,0)
> vec2 <- c(3,8,2,2,0,0,0)
```

Sin ejecutarlos, determine cuál de las siguientes declaraciones `if` tendría como resultado que una cadena se imprima en la consola. Confirma tus respuestas en R.

```
> if((vec1[1]+vec2[2])==10){ cat("Imprime el resultado!") }
```

```
> if(vec1[1]>=2&&vec2[1]>=2){ cat("Imprime el resultado!") }
```

```
> if(all((vec2-vec1)[c(2,6)]<7)){ cat("Imprime el resultado!") }
```

```
> if(!is.na(vec2[3])){ cat("Imprime el resultado!") }
```

(b) Usando `vec1` y `vec2` del ítem (a), escribe y ejecuta código que multiplica los elementos correspondientes de los dos vectores juntos si su suma es mayor que 3. De lo contrario, el código debería simplemente sumar los dos elementos.

(c) En el editor, escribe el código R que toma una matriz de caracteres cuadrada y comprueba si alguna de las cadenas de caracteres en la diagonal comienza con la letra g, minúscula o mayúscula.

Si se cumplen esta condición, estas entradas específicas se deben sobrescribir con la cadena `AQUI`. De lo contrario, toda la matriz debería reemplazarse con una matriz identidad de las mismas dimensiones.

Prueba tu código con las siguientes matrices, verificando el resultado :

```
> mymat <- matrix(as.character(1:16),4,4)
```

```
> mymat <- matrix(c("DANDELION","Hyacinthus","Gerbera",
+ "MARIGOLD","geranium","ligularia",
+ "Pachysandra","SNAPDRAGON","GLADIOLUS"),3,3)
```

```
> mymat <- matrix(c("GREAT","ejercicioss","agua","hey"),2,2,
+ byrow=T)
```

4. Analiza el siguiente código

```
> loopvec1 <- 5:7
> loopvec2 <- 9:6
> mat1 <- matrix(NA,length(loopvec1),length(loopvec2))
> mat1

     [,1] [,2] [,3] [,4]
[1,]   NA   NA   NA   NA
[2,]   NA   NA   NA   NA
[3,]   NA   NA   NA   NA
```

El siguiente bucle anidado completa `mat1` con el resultado de multiplicar cada entero en `loopvec1` por cada entero en `loopvec2`:

```
> for(i in 1:length(loopvec1)){
+   for(j in 1:length(loopvec2)){
+     mat1[i,j] <- loopvec1[i]*loopvec2[j]
+   }
+ }
> mat1

     [,1] [,2] [,3] [,4]
[1,]   45   40   35   30
[2,]   54   48   42   36
[3,]   63   56   49   42
```

(a) En aras de una codificación eficiente, reescribe el ejemplo de bucle anidado anterior, donde la matriz `mat1` se completó con los múltiplos de los elementos de `loopvec1` y `loopvec2`, utilizando solo un único bucle `for`.

(b) Usamos el comando de R

```
> switch(EXPR=cadena1,Homer=12,Marge=34,Bart=56,Lisa=78,Maggie=90, NA)
```

para devolver un número basado en el valor proporcionado de una cadena de caracteres individuales. Esta línea no funcionará si `cadena1` es un vector de caracteres. Escribe código que tomará un vector de caracteres y devolverá un vector de los valores numéricos apropiados relacionado a la cadena de caracteres. Pruéba tu respuesta con el siguiente vector:

```
> c("Peter","Homer","Lois","Stewie","Maggie","Bart")
```

(c) Supongamos que tenemos una lista llamada `unalista` que puede contener otras listas, pero supongamos que esas listas miembros de `unalista` no pueden contener listas. Escribe bucles anidados que puedan buscar cualquier `unalista` posible definida de esta manera y cuenta cuántas matrices están presentes.

Sugerencia: Configura un contador antes de comenzar los bucles que se incrementan cada vez que se encuentra una matriz, independientemente de si se trata de un miembro directo de `unalista` o si es miembro de una lista miembro de `unalista`. Verifica tus resultados:

- La respuesta es 4 si tienes lo siguiente:

```
> unalista <- list(aa=c(3.4,1),bb=matrix(1:4,2,2),
+                  cc=matrix(c(T,T,F,T,F,F),3,2),dd="cadena aqui",
+                  ee=list(c("hola","tu"),matrix(c("hola","there"))),
+                  ff=matrix(c("red","green","blue","yellow")))
```

- La respuesta es 0 si tienes lo siguiente:

```
> unalista <- list("salio algo raro",as.vector(matrix(1:6,3,2)))
```

- La respuesta es 2 si tienes lo siguiente:

```
> unalista<- list(list(1,2,3),list(c(3,2),2),list(c(1,2),matrix(c(1,2))),
+                 rbind(1:10,100:91))
```

5. (a) Para este problema, se utilizará el operador factorial. El factorial de un entero x no negativo, expresado como $x!$, se refiere a x multiplicado por el producto de todos los enteros menores que x, hasta 1. Formalmente, se escribe así:

$$\text{x factorial} = x! = x \times (x-1) \times (x-2) \times \cdots \times 1$$

Debes saber que hay un caso especial del `factorial del cero`, el cuál es 1. Esto es, $0! = 1$.

Escribe un bucle `while` que calcule y almacena como un nuevo objeto el factorial de cualquier número entero entero no negativo al decrementar `minum` por 1 en cada repetición del código. Usando este ciclo, confirma lo siguiente:

- Qué el resultado de usar `minum <- 5` es 120.
- Qué usando `minum <- 12` produce 479001600.
- El valor de `minum <- 0` devuelve correctamente 1.

(b) Considera el siguiente código, donde las operaciones dentro del ciclo `while` se han omitido

```
> unacadena <- "R fever"
> index <- 1
> ecount <- 0
> resultado <- unacadena
> while(ecount<2 && index<=nchar(mystring)){
+   # operaciones omitidas
+ }
> resultado
```

Tu tarea es completar el código para inspeccionar el objeto `unacadena` carácter por carácter hasta llegar a la segunda instancia de la letra `e` o al final de la cadena, lo que ocurra primero.

El objeto `resultado` debe ser la cadena de caracteres completa si no hay una segunda `e`, o la cadena de caracteres formada por todos los caracteres , pero sin incluir, la segunda `e` si la hay.

Por ejemplo, `una cadena <- "R fever"` debe proporcionar el resultado como `"R fev"`. Esto debe lograrse siguiendo las siguientes operaciones

- Usa `substr` para extraer el único carácter de `unacadena` en la posición `index`.
- Utiliza una verificación de igualdad para determinar si esta cadena de caracteres individuales es `"e"` OR `"E"`. Si es así, incrementa `ecount` en 1.
- A continuación, realiza una comprobación por separado para ver si `ecount` es igual a 2. Si es así, usa `substr` para establecer `resultado` igual a los caracteres entre 1 y el índice -1 .
- Incrementa `index` en 1.

6. (a) Convierta el siguiente ciclo en un ciclo implícito (utilizando alguna función de la familia `apply`) que haga exactamente lo mismo:

```
> matlist <- list(matrix(c(T,F,T,T),2,2),
+                 matrix(c("a","c","b","z","p","q"),3,2),
+                 matrix(1:8,2,4))
> matlist
> for(i in 1:length(matlist)){
+   matlist[[i]] <- t(matlist[[i]])
+   }
> matlist
```

(b) En R, almacena la matriz $4 \times 4 \times 2 \times 3$ como el objeto `qux`:

```
> qux <- array(96:1,dim=c(4,4,2,3))
```

Es decir, es una matriz tetradimensional compuesta de tres bloques, siendo cada bloque una matriz compuesta por dos capas de matrices de 4×4. Luego, haz lo siguiente:

- Escribe un bucle implícito que obtenga los elementos diagonales de todas las matrices de segunda capa para producir la siguiente matriz:

```
     [,1] [,2] [,3]
[1,]   80   48   16
[2,]   75   43   11
[3,]   70   38    6
[4,]   65   33    1
```

- Escribe un bucle implícito que devuelva las dimensiones de cada una de las tres matrices formadas accediendo a la cuarta columna de cada matriz en `qux`, independientemente de la capa o bloque, enlazado por otro bucle implícito que encuentre la suma de filas de esa estructura, resultando en el siguiente vector:

```
[1] 12 6
```

7. (a) A está en una avenida, la cual puede ser representada como una recta numérica entera. A y B han quedado en que A se encargaría de arreglar la posición de la antena de radio de toda la avenida, de forma de que esta quede en una posición perfecta. Sin embargo, A últimamente no ha dormido lo suficiente, por lo que está en `modo zombie`, recibiendo los datos de B pero sin recordar todo por completo o sin si quiera apuntándolos en orden. Dado que la información de B es de una cadena de caracteres que consiste de:

   - $+$, que significa ir de la posición x actual a la $x+1$
   - $-$, que significa ir de la posición x actual a la $x-1$

   Además, la cadena de A tiene un ? en los datos que no recuerda correctamente.
   Dada la cadena de información original y la cadena que A cree recibir, calcula la probabilidad de que A coloque satisfactoriamente la antena y dar la respuesta con 9 decimales.

   Restricciones:

   - $1 \leq |\text{Info}| \leq 10$
   - $1 \leq |A| \leq 10$
   - $|\texttt{A}| = |\texttt{Info}|$

   Donde $|S|$ expresa la longitud de la cadena S.

   Ejemplos

   - Entrada 1

     ```
     ++-+-

     +-+-+
     ```

   - Salida 1
     1.000000000

   Explicación 1
   A recibe una permutación de la cadena de información, por lo que si colocara la antena en la posición correcta.

   Ejemplos

   - Entrada 2

     ```
     +-+-

     +-??
     ```

   - Salida 2
     0.500000000

   Explicación 2
   Los datos que A no recuerda deben ser un $+$ y un $-$ sin importar el orden, dando que solamente con $+-$ o $-+$ se logra colocar bien la antena, ante el total de $2^2 = 4$ eventos posibles.

   (b) Jessica, Vilma y Esther están viajando a Uganda con su padre. Antes de viajar, las 3 comienzan a discutir por el asiento del copiloto. Para arreglar la riña, decidieron que cada uno lanzaría 3 dados, calcularían la suma de sus resultados y la que tuviera mayor suma ganaría el derecho de sentarse al lado de su padre. Vilma obtuvo B puntos y Esther C, por lo que Jessica quiere saber la probabilidad de que gane el encuentro. Debido a que Jessica es la menor de las 3, si obtiene un puntaje mayor o igual a $max(B, C)$ la dejaran ganar. Expresa la probabilidad como un número con el formato x/y, la cual es una fracción irreductible.

   Ejemplo

   - Entrada 1

```
4 4
```

- Salida 1 215/216

Explicación
La única situación en la que Jessica no gana es cuando saca 3 puntos, situación que se logra con los 3 dados dando 1 , a comparación de los 216 posibles eventos, por lo que gana el resto.

8. La idea de inversión de matriz, es posible sólo para ciertas matrices cuadradas (aquellas con un número igual de columnas como filas). Estas inversiones se pueden calcular utilizando la función `solve`, por ejemplo:

```
> solve(matrix(1:4,2,2))

     [,1] [,2]
[1,]   -2  1.5
[2,]    1 -0.5
```

Ten en cuenta que `solve` muestra un error si la matriz dada no se puede invertir. Con esto en mente, escribe una función en R que intente invertir cada matriz en una lista, de acuerdo con las siguientes pautas:

- La función debería tomar cuatro argumentos
  - La lista x cuyos elementos deben probarse para la inversión de matriz.
  - Un valor `noinv` para completar los resultados donde los elementos de la matriz en x no pueda invertirse, por defecto un NA.
  - Una cadena de caracteres `nonmat` será el resultado si un elemento dado de x no es una matriz, por defecto a escribiendo `"no es una matriz!"`.
  - Un valor lógico `silent`, predeterminado en `TRUE`, que se pasará a `try` en el cuerpo del código.
- La función debería verificar primero si x es, de hecho, una lista. Si no, debería lanzar un error con un mensaje apropiado.
- Entonces, la función debería garantizar que x tenga al menos un elemento. Si no, debería lanzar un error con un mensaje apropiado.
- Luego, la función debería verificar si `nonmat` es una cadena de caracteres. De lo contrario, debería intentar coerzar a una cadena de caracteres y debería emitir una advertencia apropiada.
- Después de estas comprobaciones, un bucle debe buscar cada miembro i de la lista x.
  - Si el elemento i es una matriz, trata de realizar una inversión con `try`. Si es invertible sin error, sobrescribe el elemento i de x con el resultado. Si se detecta un error, entonces el elemento i de x debe sobrescribirse con el valor de `noninv`.
  - Si el elemento i no es una matriz, entonces el elemento de i de x debe sobrescribirse con el valor de `nonmat`.
- Finalmente, se debe devolver la lista x modificada.

Ahora, prueba la función usando los siguientes valores de argumento para asegurarse de que la respuesta sea la esperada:

- x como

```
> list(1:4,matrix(1:4,1,4),matrix(1:4,4,1),matrix(1:4,2,2))
```

y todos los demás argumentos por defecto.

- x como el ítem anterior, `noninv` como `Inf`, `nonmat` como 666, `silent` por defecto.

- Como en el ítem anterior, pero ahora con `silent =FALSE`.
- *x* como

```
> list(diag(9),matrix(c(0.2,0.4,0.2,0.1,0.1,0.2),3,3),
+      rbind(c(5,5,1,2),c(2,2,1,8),c(6,1,5,5),c(1,0,2,0)),
+      matrix(1:6,2,3),cbind(c(3,5),c(6,5)),as.vector(diag(2)))
```

y `noinv` como `"matriz inadecuada"`; todos los otros valores por defecto.

Finalmente, prueba los mensajes de error intentando llamadas a la función conseguida con las siguientes expresiones:

- *x* como "hola"
- *x* como `list()`