# Ejercicios de programación II: Fundamentos

**[IMSER 2012]**

---

## Instrucciones generales:

### Archivos incluidos:

El archivo con los ejercicios del práctico debe bajarse y descomprimirse en disco duro, creando la carpeta `rep-2` (nota: no debe dentro de ningún disco, partición o carpeta protegida a la escritura, como puede ser un disco duro externo de backup). Usted deberá abrir el RStudio y seleccionar dicha carpeta como su directorio de trabajo con `setwd` o en RStudio la combinación **Ctrl + Shift + K**. En esta carpeta se encuentran algunos archivos que usted deberá modificar:

- `aprobados.R`
- `aprobados2.R`
- `mejorcitos.R`
- `franjas.R`
- `data.frame.R`
- `ordenacion.R`
- `lista.R`
- `print.listaCalif.R`

Adicionalmente los siguientes archivos son necesarios, pero **no deben ser modificados** para que el método de calificación automático funcione correctamente.

- `evaluar.R`
- `notas.csv`
- `datos`
- `INSTRUCCIONES.pdf`
- `calificaciones.R`
- `ej2.RData`

## Mecanismo de corrección:

Nota: más recomendaciones **importantes** se hacen en el documento Dinámica de los repartidos.

Lo primero que debe hacer es cargar el archivo evaluar.R con la función `source` y la codificación de caracteres "UTF-8" (lo cual afecta a la función `evaluar` en particular), de la siguiente manera:

```
options(encoding = "utf-8")
source("evaluar.R")
```

Si usted ha ejecutado todos los pasos anteriores correctamente, la siguiente frase debería verse en la consola:

```
Archivo de codigo fuente cargado correctamente
```

En caso de que ocurra un error o se vea otro mensaje en la consola, verifique que los archivos se descomprimieron correctamente y que usted está trabajando en la carpeta correspondiente con el comando `getwd()`.

Usted trabajará modificando los contenidos de dichos archivos con RStudio (u otro programa de su preferencia) según las consignas que se describen a continuación. Luego de terminar cada ejercicio y **guardando el archivo** correspondiente en el disco duro, usted podrá verificar rápidamente si su respuesta es correcta ejecutando el comando:

```
evaluar()
```

y además podrá en todo momento verificar su puntaje con la función `verNotas()`. Tenga siempre en cuenta que, a **menos que sea indicado** por la letra del ejercicio, las soluciones deben ser genéricas y por lo tanto deben servir aún si se modifican los datos originales (i.e.: no use valores fijos si no comandos). Usualmente se utilizan valores generados de forma aleatoria para las correcciones automáticas. Los objetos que son evaluados en la corrección automática estarán indicados con un asterísco en las instrucciones de cada script. Nótese además que en los archivos **se indica claramente en dónde se inicia y dónde finaliza su código** y que debe respetar esta organización para que la corrección de los ejercicios funcione bien.

### Al finalizar

Una vez terminados y guardados los archivos de los ejercicios del repartido, usted deberá ejecutar `evaluar()` y seleccionar la última opción ("Todos") y

luego subir el archivo "datos" (sin extensión), incluido en la carpeta "rep-1", a la sección de entregas de la portada del curso en la plataforma EVA. Este archivo se podrá reemplazar con uno más nuevo, en caso de que desee corregir algún error; en caso de querer que el archivo sea corregido antes de la fecha de entrega, puede cambiarle el nombre a "datos-finalizado", pero en ese caso la nota no se cambiará de ahí en adelante.

**Código de Honor**

Si bien animamos a que trabaje en equipos y que haya un intercambio fluido en los foros del curso, es fundamental que las respuestas a los cuestionarios y ejercicios de programación sean fruto del trabajo individual. En particular, consideramos necesario que no utilize el código creado por sus compañeros, si no que debe programar sus propias instrucciones, ya que de lo contrario supone un sabotaje a su propio proceso de aprendizaje. Esto implica también evitar, en la medida de lo posible, exponer el código propio a sus colegas. Como profesores estamos comprometidos a dar nuestro mayor esfuerzo para dar las herramientas y explicaciones adecuadas a fin de que pueda encontrar su propio camino para resolver los ejercicios.

En casos de planteos de dudas a través del foro, en los que considere que es imposible expresar un problema sin exponer su própio código, entonces es aceptable hacerlo. De todas formas en estos casos es preferible que envíe su código por correo electrónico directamente a un profesor, explicando la problemática.

---

## 1. Datos de calificaciones

Con el archivo "calificaciones.R" se generan calificaciones ficticias de estudiantes de secundaria, utilizando rutinas de generación de números aleatorios. Lo primero que debe hacer entonces es ejecutar el comando

```
source("calificaciones.R")
```

para generar los vectores `cal` y `gen`, los cuales deberían encontrarse en su área de trabajo (o "Workspace" en RStudio). En el primero se encuentran las notas de dichos estudiantes, con valores que van del 1 al 12, mientras que el segundo indica el sexo de cada uno de ellos (codificados como "V" y "M", varón y mujer respectivamente). Nótese que las clases de ambos vectores son diferentes, siendo uno numérico y el otro caracter. Ejecute los siguientes comandos para hechar un vistazo rápido a los datos:

```r
# Histograma de las notas:
hist(cal)
# Cantidad de mujeres y de varones:
table(gen)
```

### 1.a Porcentaje de aprobados

Script: "aprobados.R"

Considerando que los aprobados son todos aquellos que tienen una nota de 5 o mayor, usted deberá cuantificar el porcentaje de aprobaciones (*no confundir con fracción*). En el archivo `aprobados.R` deberá escribir el código necesario para calcular este valor. Tenga en cuenta que para esta tarea necesitará usar *operadores relacionales*. Existen varias formas de obtener el valor final, por lo que queda a su gusto determinar el camino a tomar.

Recuerde que su solución debe ser genérica y por lo tanto funcionar con cualquier vector `cal` posible. El comando `cal <- rpois(10, 6.5)` genera un nuevo vector `cal` aleatorio de 10 elementos que puede usar para verificar que su solución es correcta.

### 1.b Aprobados por género

Script: "aprobados2.R"

El objetivo aquí es determinar la cantidad y porcentaje de aprobados para varones y mujeres por separado. Es decir, se debe determinar la cantidad de varones aprobados en relación a la cantidad de varones totales, y lo mismo para las mujeres. El código para ejecutar esta tarea se debe escribir en el archivo "aprobados2.R".

La estrategia más sencilla es primero separar las calificaciones de varones y mujeres en dos vectores (`v` y `m` por ejemplo) y luego aplicar el método usado en el ejercicio anterior en ambos. Para esto evidentemente tendrá que usar el vector `gen` además de operadores relacionales, en particular `==` (o su contraparte `!=`).

Recuerde que su solución debe ser genérica.

### 1.c Los mejorcitos

Script: "mejorcitos.R"

En esta parte lo que hará es seleccionar al grupo de los mejores calificados, trabajando con el script "mejorcitos.R". El objetivo es crear un vector numérico llamado `mejores`, el que tendrá las calificaciones del 25% de los estudiantes con mejor nota. Es decir, en caso de que fueran 100 valores, queremos tomar los

25 más altos. Para esto una estrategia simple es ordenar los valores de menor a mayor y luego elegir los últimos elementos del vector resultante. Lo primero será entonces crear un vector llamado `ord` con la función `sort`. Posteriormente es necesario determinar la iésima posición del mismo a partir de la cual se deben tomar los valores, para lo cual seguirá la ecuación:

$$i = \lceil n \cdot 0.75 \rceil$$

donde $n$ es el número de elementos de `cal` y se utiliza la función techo: es decir $\lceil x \rceil$ indica *el siguiente valor entero más alto* que $x$. Entonces el vector `mejores` consistirá en los elementos de `ord` entre las posiciones $i$ (inclusive) y $n$. Puede buscar la función techo en R con el comando `??ceil`.

Nota: el `i` hallado debería cumplir que al evaluar

```
i/length(cal)
```

el resultado es un número muy cercano a 0.75, *pero nunca menor*. Finalmente puede visualizar cómo se distribuyen estos datos usando las funciones `table` o `hist`.

### 1.d Franjas de notas

Script: "franjas.R"

El objetivo de esta sección es crear un nuevo vector de clase "character" el cual tendrá las letras "A", "B", "C" y "D" a fin de indicar cuatro franjas de notas, como se describe a continuación:

- A: $x \leq 3$
- B: $3 < x \leq 6$
- C: $6 < x \leq 9$
- D: $9 < x$

siendo $x$ la nota del estudiante. El nuevo vector character se llamará `ctg`, tendrá la misma cantidad de elementos que número de calificaciones y para cada posición tendrá asignada la letra correspondiente (en mayúsculas). Por ejemplo, para el siguiente vector `cal`:

```
> cal
[1]  2 11  3  6  7  6  9 5
```

El vector `ctg` correspndiente es:

```
> ctg
[1] "A" "D" "A" "B" "C" "B" "C" "B"
```

Finalmente deberá obtener la cantidad de casos para cada franja (en el ejemplo anterior serían 2, 3, 2, 1 para A, B, C y D respectivamente) y guardar estos valores en el vector `conteo` (las funciones `sum` o `table` pueden ser de utilidad aquí). Los nombres de los elementos de dicho vector deben ser A, B, C y D (en ese orden y respetando mayúsculas).

El archivo para esta tarea es "franjas.R".

**Sugerencia:** Una forma sencilla de hacer `ctg` es crear un vector del tipo "character" inicial, con funciones tales como `rep` o `character`, para luego modificarlo, utilizando el esquema:

```
ctg <- character(10)
ctg[c(4, 7)] <- "Algo"
```

Probablemente deba usar operadores lógicos y relacionales para reproducir las 4 condiciones que definen las franjas.

## 2. Organización de los datos

En esta sección se crearán data.frames y listas en base a datos de calificaciones similares a los del ejercicio 1. Para esto puede continuar utilizando los objetos creados en dicho ejercicio, o alternativamente cargar objetos ya preparados para este ejercicio con el comando:

```
# Comando opcional:
load("ej2.RData")
```

### 2.a Crear una data.frame

Sript: "data.frame.R"

En esta sección deberá modificar el archivo "data.frame.R" para crear un objeto de la clase "data.frame" llamado `datos.calif`, cuyas columnas/variables sean los vectores `cal`, `gen` y `ctg`. Los nombres de tales columnas en la data.frame serán "nota", "genero" y "franja" (respetando mayúsculas y minúsculas). Utilice su método de preferencia para generar dicha data.frame. Una vez hecho, el siguiente comando debería dar un resultado similar a este (con distintos valores):

```
> head(datos.calif)
  nota genero franja
1    9      M      C
2    8      V      C
3    5      M      B
4   11      V      D
5   11      V      D
6    5      M      B
```

Nota: siguiendo al menos una de las posibles formas de crear esta data.frame, es posible que los vectores character (`gen` y `ctg`) sean coercionados a factor por R al crear la data.frame (ver `class(datos.calif$genero)`.

### 2.b Ordenar la tabla

Script: "ordenacion.R"

el objetivo aquí es modificar la tabla `datos.calif` creada en la sección anterior, de forma que las filas estén ordenadas en función de la nota (de menor a mayor). Una vez terminado, las primeras filas de `datos.calif` se deberían ver similares al siguiente ejemplo:

```
> head(datos.calif)
    nota genero franja
40     1      V      A
178    1      M      A
181    1      M      A
194    1      V      A
236    1      V      A
246    1      M      A
```

y las últimas parecidas a esto:

```
> tail(datos.calif)
    nota genero franja
193   11      V      D
242   11      M      D
252   11      M      D
272   11      M      D
135   12      V      D
136   12      V      D
```

**Sugerencias** Para este ejercicio es necesario comprender el funcionamiento de la función `order`, el uso de los índices en vectores y/o matrices, así como el de los corchetes y/o el operador `$`. Nótese además que una expresión de la forma

```
x <- x[i]
```

es perfectamente válida y que el objeto `x` es sobreescrito en el proceso (y esto sirve para vectores, como `x`, matrices, data.frames, etc, *haciendo los cambios pertinentes*).

Nota 1: la función `order` puede ser bastante contraintuitiva. En la edición anterior del curso algunos estudiantes encontraron este link muy útil para entenderla. Una definición provisoria y mínima, pero útil es: `sort(a) == a[order(a)]`.

Nota 2: usando la función `arrange` el paquete `plyr` puede verificar que su data.frame está correctamente ordenada. El siguiente código sirve de ejemplo (la única diferencia es que los *nombres de las filas* no se conservan al usar `arrange`):

```
install.package("plyr")
library(plyr)
arrange(datos.calif, nota)
```

### 2.c Una lista con los datos

Script: "lista.R"

Vamos a crear ahora una lista con los datos generados en ejercicios anteriores (los puede cargar del archivo "ej2.RData" si es necesario), modificando el código del archivo "lista.R". Esta lista se llamará `analisis.calif` y tendrá los siguientes componentes (preste atención a los nombres):

1. tabla: la data.frame creada en el ejercicio 2.a (y modificada en 2.b).
2. conteo: el vector nombrado con los conteos por franjas de calificaciones.
3. aprob: una lista con 3 elementos:
   - atot: porcentaje de aprobación total (`p.apr`)
   - avar: porcentaje de aprobación de varones (`p.apr.v`)
   - amuj: porcentaje de aprobación de mujeres (`p.apr.m`)

Si su lista ha sido construida correctamente, al usar la función `str` debería ver algo similar a lo siguiente (con la salvedad de que los numeros serán diferentes):

```
> str(analisis.calif)
List of 3
 $ tabla :'data.frame':  272 obs. of  3 variables:
  ..$ nota  : num [1:272] 1 1 1 1 1 1 2 2 2 2 ...
  ..$ genero: Factor w/ 2 levels "M","V": 2 1 1 2 2 1 2 1 1 1 ...
  ..$ franja: Factor w/ 4 levels "A","B","C","D": 1 1 1 1 1 1 1 1 1 1 ...
 $ conteo: Named num [1:4] 38 113 99 22
  ..- attr(*, "names")= chr [1:4] "A" "B" "C" "D"
 $ aprob :List of 3
  ..$ atot: num 71
  ..$ avar: num 73.4
  ..$ amuj: num 68.2
```

### 2.d Extra: una clase nueva y un método asociado

(*Este ejercicio es opcional, aunque puede sumar puntos en su calificación final del repartido*)

Script: "print.listaCalif.R"

Cuando se manejan estructuras de información sofisticadas, como la lista creada en la sección anterior, no es mala idea definir una nueva clase y algunos métodos asociados para hacer el trabajo más fluido. Esto es útil en particular cuando es una tarea que se repite muchas veces a lo largo de un proyecto o en trabajos de rutina.

Actualmente el objeto `analisis.calif` debería ser de la clase "list", lo que podemos comprobar con el comando:

```
> class(analisis.calif)
[1] "list"
```

Hagamos una nueva clase llamada "listaCalif" la cual va a estar compuestas por listas con la estructura de `analisis.calif`. Lo primero que vamos a hacer es cambiar la clase de este objeto de la siguiente manera:

```
> class(analisis.calif) <- "listaCalif"
```

Esto por ahora no traerá mayores cambios, pero una vez que usted modifique y ejecute el archivo "print.listaCalif.R" se habrá definido un nuevo método de `print` para esta nueva clase. Para entender la diferencia que esto hace, veamos lo que sucede cuando se escribe el nombre del objeto `analisis.calif` en la consola:

```
> analisis.calif
...(muchos números)
```

R simplemente arroja todos los datos de la lista. Sin embargo, al cargar "print.listaCalif.R", se puede ver algo así:

```
> source("print.listaCalif.R")
> analisis.calif
Porcentaje total de aprobaciones: 70.96 %
  En varones: 73.43 %
  En mujeres: 68.22 %
La nota promedio fue de: NA
  En varones: NA
  En mujeres: NA
Conteos por franja de nota:
  1--3   4--6   7--9 10--12
    38    113     99     22
```

¡Esta es la gran utilidad de definir métodos para una clase en R! En este caso, en lugar de imprimir en pantalla una cantidad de números difícles de leer, ahora se pueden ver los resultados más relevantes sin todo ese ruido. Nótese que esto sólo es posible gracias a que `print` es una función genérica y por lo tanto permite hacer nuevos métodos en cualquier momento. No muchas funciones pertenecen a esta categoría (ver `?GenericFunctions` por mayor información).

El único problema en este caso es que no se calculan bien los promedios de las notas y por lo tanto se ven unos "NA" en la salida de la consola. El objetivo de este ejercicio es modificar la función `print.listaCalif` para que calcule estos 3 promedios y así los imprima correctamente cada vez que se llama a un objeto de la nueva clase "listaCalif".

Invitamos además a mirar el resto del código de la función `print.listaCalif` para tratar de entender los pasos que se toman para generar esta salida.

**Consejos y sugerencias** Nótese el uso del operador `$` en varios comandos internos de dicha función, ya que usted deberá usarlos también para calcular los promedios.

Debe saber también que "adentro" de la función existe un ambiente distinto al de sus sesión de trabajo. En la práctica esto implica que todo lo que está fuera de la función, en principio "no existe" (aunque esto no es tan así, veremos en la unidad 5). Por lo tanto *cuidese de no utilizar objetos que no estén definidos dentro de la función o en la lista de argumentos* (lista que en este caso es de un sólo elemento, `x`). Esto es particularmente importante en este ejercicio debido a que esta función debe pasar por el mecanismo de corrección del curso.

El siguiente busca ser un ejemplo ilustrativo:

```
# Incorrecto:
a <- rnorm(8)
```

```r
f <- function(x) {
    a * 3 + 5
}
# Correcto:
a <- rnorm(8)
g <- function(x) {
    x * 3 + 5
}
f(a)  # Resultado correcto
f(6)  # Resultado incorrecto
g(a)  # Resultado correcto
g(6)  # Resultado correcto
```

(Podrá notar que la función `f` falla por su incapacidad de ser una solución genérica.)