# Ejercicios de programación I: Introducción

**[IMSER 2013]**

---

## Instrucciones generales:

### Archivos incluidos:

El archivo con los ejercicios del práctico debe bajarse y descomprimirse dentro de la carpeta del curso, creando la subcarpeta `rep-1`. 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:

- `hipot.R`
- `areaMax.R`
- `dist.R`
- `varianza.R`
- `zenon.R`
- `geom.R`
- `shannon-1.R`
- `shannon-2.R`

Cada uno de estos archivos se corresponde con un ejercicio del repartido. Se trata de archivos de texto plano (como un `.txt`, pero con una extensión diferente). En cada archivo se indica con precisión en dónde debe usted escribir su código (o modificar lo que ya está escrito). Para que el sistema de corrección funcione bien, **no cambie** el texto que se encuentra por fuera, incluyendo las que se usan para indicar el inicio y el final del mismo. Nótese además que **esto no impide** ampliar o reducir el espacio que usted tiene para escribir, subiendo o bajando cualquiera de las "líneas límite" del archivo.

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

- `evaluar.R`
- `datos`
- `notas.csv`
- `INSTRUCCIONES.pdf`

## 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`:

```
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. Geometría

Como sabrá probablemente, el filósofo griego Pitágoras es el supuesto autor del teorema homónimo, el cual sirve para calcular la hipotenusa de un triángulo rectángulo. Este cálculo se realiza con la fórmula $h^2 = a^2 + b^2$, en donde $h$ es la hipotenusa y $a$ y $b$ los catetos.

### 1.a Funciones simples

Script: hipot.R

El siguiente código sirve para crear la función `hipot`, la cual acepta como argumentos los anchos de los catetos y calcula el valor de $h$ correspondientes:

```
hipot <- function(cat.ad, cat.op) {
    out <- sqrt(cat.ad^2 + cat.op^2)
    out
}
```

Nótese que los objetos* `cat.ad` y `cat.op` son los "argumentos" de la función (y eso lo sabemos porque están dentro del paréntesis que sigue a la función especial `function`). En las líneas "internas" (i.e.: aquellas que están entre el { inicial y el } final) de las funciones que haremos en este repartido usaremos, por ahora, sólamente estos objetos para hacer todos los cálculos requeridos.

(*: estrictamente no son objetos, en el sentido de objetos existentes en nuestra sesión; de todas formas representan objetos que existen temporalmente en el momento en que la función es evaluada. Veremos más sobre este tema en la unidad 6)

Tomando este caso como ejemplo, complete el código del script "hipot.R" para hacer las funciones `area` y `co` tales sean capaces de calcular el área de un triángulo y el cateto opuesto respectivamente. Las entradas para la función `area`, es decir, sus argumentos, serán los anchos de los catetos (tal como el la función `hipot`, incluyendo el órden). Para el caso de `co` los argumentos serán los anchos del cateto adyacente y de la hipotenusa, **en ese orden**. Recuerde que para ejecutar este script debe usar la función `source` de la siguiente manera:

```
source("hipot.R")
```

o usar los atajos de teclado de RStudio. Luego en la sesión de trabajo aparecerán los distintos objetos definidos en el script (verifíquelo con `ls()`), incluyendo a las funciones `area` y `co`.

Si sus funciones están correctas, usted debería obtener estos resultados:

```
area(3, 4)
```

```
## [1] 6
```

```
co(3, 5)
```

```
## [1] 4
```

Pero también recuerde que estas funciones deben hacer el cálculo correcto *para cualquier par de valores* de catetos e hipotenusas, es decir deben ser ***genéricas*** (con la restricción de que la hipotenusa siempre es mayor que los catetos).

Nota: dado que para obtener el cateto se hace una resta, a cuyo resultado se debe hallar la raíz cuadrada, considere que debe evitar valores negativos. La forma más básica de evitarlo es encontrando el valor absoluto, usando la función `abs`.

### 1.b Área máxima

Script: areaMax.R

En la lección 1.2 vimos cómo con un loop podemos ver fácilemnte todos los valores posibles de salida de la función `prp`. Ahora queremos hacer algo similar con la función `area`. Es decir, si el cateto adyacente $x$ puede tomar cualquier valor tal que $0 < x < h$ (siendo $h$ la hipotenusa), entonces podemos buscar cuál es el área máxima posible y para qué valor de $x$ se da. Obviamente este resultado se puede encontrar analíticamente, pero para los propósitos del ejercicio es útil calcularlo numéricamente.

Siguiendo el ejemplo de la lección 1.2 entonces el siguiente código serviría para encontrar esa área (suponiendo que las funciones `area` y `co` están correctamente definidas):

```
hip <- 10
cat.ad <- seq(0.001, hip - 0.001, by = 0.1)
for (i in 1:length(cat.ad)) {
    cat.op <- co(cat.ad[i], hip)
    cat("ancho del cateto ad.:", cat.ad[i], "\n")
    cat(">> area:", area(cat.ad[i], cat.op), "\n")
}
```

Si usted ejecuta este código, podrá ver que el área máxima está cercana a 25 y que el ancho del cateto adyacente correspondiente es aproximadamente 7.1 (si ejecuta el script "ejTriang.R" puede visualizar todos estos triángulos). Pero este es un método sumamente engorroso en comparación a lo aque se puede hacer en R. Para empezar, como se menciona en la lección 1.2, en su versión escrita la menos, el cálculo de todas estas áreas (y de los catetos opuestos asociados) se puede hacer sin loops. Esta capacidad de R es muy importante para aumentar la velocidad de ejecución de ciertas tareas y se denomina **vectorización**.

El siguiente es un ejemplo de vectorización, ya que calcula todos los valores posibles de `cat.op` y `a` sin utilizar un **loop** (es decir, no utiliza el comando especial `for` mostrado anteriormente). Para el caso que las áreas, ejecute el siguiente código:

```
hip <- 10  # Hipotenusa
cat.ad <- seq(0.001, hip - 0.001, by = 0.1)  # Valores del cateto adyacente
cat.op <- co(cat.ad, hip)  # Cálculo de los catetos opuestos
a <- area(cat.ad, cat.op)  # Cálculo de las áreas
```

De esta forma tiene un vector `a` con todos los valores de áreas. Nótese el uso de la función `seq` para crear una secuencia de valores entre 0.001 y $h - 0.001$, utilizados como anchos de los distintos catetos adyacentes. De esta forma sólo

quedaría buscar cuál de los valores de área es el mayor y determinar en qué posición se encuentra. Esto lo deberá hacer usted, completando el código en el archivo "areaMax.R". Para ello recomendamos considerar dos funciones de R:

- Las funciones `which` o `which.max` se pueden utilizar para encontrar la posición (i.e.: un número entero positivo) del valor máximo dentro de un vector (siendo la primera una opción más versatil en verdad). Utilice la documentación para aprender más de estas funciones y así encontrar la posición del valor máximo dentro del vector `a`.
- El uso de los corchetes `[` y `]` sirve para seleccionar/modificar el i-ésimo elemento dentro de un vector (y también se usa con matrices). Acceda a la documentación del uso de estos operadores con el comando `?"["`

Cuando complete el código podrá confirmar que ha encontrado verdaderamente el valor máximo con los comandos:

```
plot(cat.ad, a, type = "l", ylab = "Área", main = "Área máxima: línea roja")
abline(v = cat.ad[i], h = amax, col = "red", lwd = 1.3, lty = 3)
```

Adicionalmente, si su solución es realmente genérica entonces el resultado del siguiente código debe terminar en `TRUE`:

```
hip <- rnorm(1, 10)
cat.ad <- runif(100, 0.1, hip - 0.1)
cat.op <- co(cat.ad, hip)
a <- area(cat.ad, cat.op)
source("areaMax.R")
max(a) == amax  # ¿TRUE or FALSE?
```

(Lo que hace este código es crear valores aleatorios de los objetos `hip`, `cat.ad`, `cat.op` y `a` y verificar que su código funciona con estos.)

### 1.c Distancias entre puntos

Script: dist.R

La definición de distancia euclidiana se basa en el teorema de pitágoras. Para dos puntos en un espacio bidimensional (un plano) la distancia entre ambos se puede calcular a través de las coordenadas cartesianas de los mismos, tal como lo muestra la Figura 1.

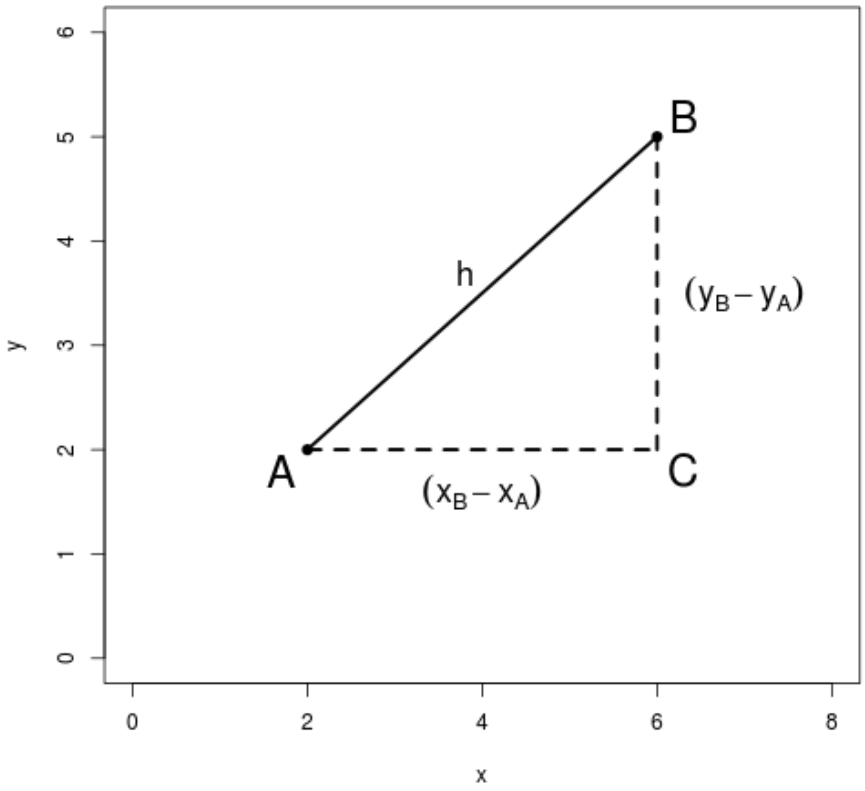


Figure 1: triángulo ABC

**Figura 1:** Los puntos A, B y C de coordenadas $(x_A, y_A)$, $(x_B, y_B)$ y $(x_B, y_A)$ pueden formar un triángulo rectángulo como muestra la figura, cuyos catetos tienen los valores indicados entre paréntesis. Por lo tanto, la distancia entre A y B es la hipotenusa del triángulo y se puede hallar con la fórmula de Pitágoras. Nota: puede reproducir este gráfico con la función `plotTriang` contenida en el script "plotTriang.R"; además lo invitamos a que examine dicho código si le interesa entender cómo se crea tal figura.

---

En el presente ejercicio vamos a utilizar este concepto, así como las recién aprendidas funciones `which` y `which.max`, y también `which.min`. La idea es encontrar distancias entre puntos y determinar distancias mínimas y máximas. En el archivo "dist.R" el código inicial (8 líneas) debería reproducir un gráfico muy similar a la **Figura 2**.

Los objetos `coorx` y `coory` son las coordenadas en los ejes x e y respectivamente de todos los puntos blancos. Estos fueron obtenidos con un generador de números aleatorios. El punto negro en el centro de la figura es nuestro foco de interés.

Para encontrar cuáles son el más cercano y el más lejano del punto negro primero debemos calcular las distancias desde éste a todos los demás. Esto lo podemos hacer usando la función `hipot` creada anteriormente. Para esto debemos calcular, con los valores de las coordenadas, los "catetos", usando de referencia el método descripto en la **Figura 1**. Para esto usted necesita saber que las coordenadas del punto central son (0.5, 0.5).

Para comprobar que su resultado es correcto, puede utilizar el siguiente código:

```
plot(coorx, coory)
points(0.5, 0.5, pch = 19)
points(coorx[i], coory[i], pch = 19, col = "red")
points(coorx[j], coory[j], pch = 19, col = "darkgreen")
```

Complete el código del archivo "dist.R" para completar esta parte del ejercicio y recuerde que su solución debe servir sin importar la ubicación exacta de los puntos en el plano.

## 2. Cálculo de sumatorias

Crear el código necesario para calcular sumatorias en R u otros lenguajes no es intiutivo cuando no se tiene experiencia en programación. En R la función `sum` sirve para este propósito. Si tenemos un vector `s` con una cantidad arbitraria de elementos, entonces la suma de todos ellos se realiza con el comando `sum(s)`. Por ejemplo:

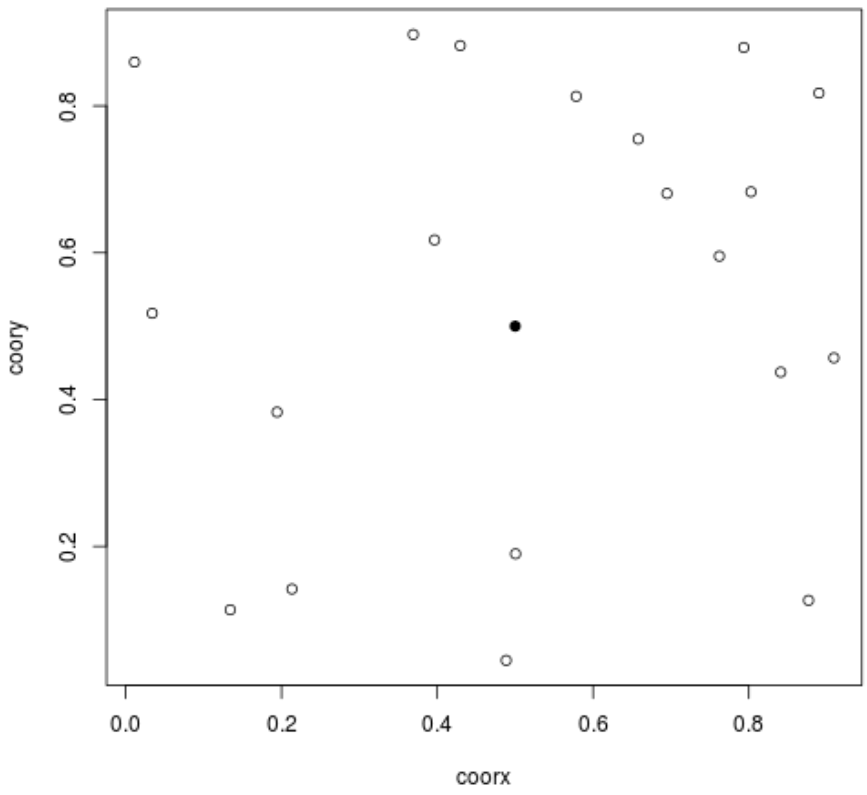


Figure 2: puntos en el plano

```r
s <- c(-10, 5, 6)
sum(s)
```

```
## [1] 1
```

Nota: la función `c` aquí es usada para crear `s`, un vector que es simplemente la secuencia de los números -10, 5, 6. Dicha función es una de las más usadas en R.

Sin duda que este es el caso más trivial. Cuando queremos trabajar con algún caso más elaborado, es conveniente entonces separar el problema en dos partes, (1) Crear el vector `s`, el cual contiene todos los términos de la sumatoria y (2) ejecutar la función `sum`, y es por eso que así lo hicimos en el ejemplo. Por supuesto que en la primer etapa es en donde se hace casi todo el trabajo. Muchas veces nos encontramos con sumatorias que están definidas con funciones matemáticas. Un ejemplo es la fórmula para calcular la varianza no sesgada de una muestra de `n` datos:

$$\sigma^2 = \frac{1}{n-1} \cdot \sum_{i=1}^{i=n} (x_i - \overline{x})^2$$

En donde $\overline{x}$ denota el valor promedio de la muestra de datos, calculado como $\frac{1}{n} \cdot \sum_i x_i$. Llamaremos "término de la sumatoria" a la función está incluida en la misma, es decir $(x_i - \overline{x})^2$.

### 2.a Cálculo de la varianza de una muestra

Script: varianza.R

En el archivo "varianza.R" usted deberá escribir los pasos necesarios para calcular la varianza del vector de ejemplo `x` que se muestra en el archivo (siga las instrucciones incluidas en los comentarios).

Si su solución es correcta, lo cual implica que es genérica, entonces el valor de `out` obtenido coincidirá con la salida de la función `var`. Puede correr las siguientes líneas varias veces para determinar si esto es así:

```r
source("varianza.R")
out == var(x)
```

Esto producirá un `TRUE` o un `FALSE` en caso de que `out` esté bien o mal calculado, respectivamente.

### 2.b Paradoja de Zenón

Script: zenon.R

Según la clásica paradoja de la dicotomía de Zenón es imposible caminar de un punto A a un punto B, debido a que primero debemos movernos la mitad del camino, posteriormente avanzar la mitad de la mitad del camino y así ad infinitum, sin llegar jamás a B. Esto se traduce en avanzar primero 1/2 de camino, luego 1/4, luego 1/8, luego 1/16 y así sucesivamente. Entonces, luego de $n$ pasos de este tipo se alcanza una fracción de camino equivalente a:

$$Z_n = \sum_{i=1}^{i=n} \frac{1}{2^i} = \frac{1}{2^1} + \frac{1}{2^2} + \frac{1}{2^3} + ... + \frac{1}{2^n} = \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + ... + \frac{1}{2^n}$$

(¡nótese que el primer valor de $i$ es 1 y no 0!)

Si es cierta la proposición de Zenón, entonces no importa que tan grande sea $n$, esta sumatoria siempre será menor a 1, a pesar ser este el límite de la misma cuando $n \to \infty$. Podemos verificar numéricamente esta afirmación usando R: si elegimos un valor $\varepsilon$ arbitrariamente pequeño, entonces podemos encontrar un $n$ tal que $1 - Z_n < \varepsilon$.

En este ejercicio, usted deberá completar el código del archivo "zenon.R" según las instrucciones que se indican en el mismo. Para que usted pueda tener una referencia, considere que si el código se hace correctamente, la sumatoria entonces para $n = 3$ el valor `out` debería ser 0.875.

Además usted debe probar distintos valores del entero `n`, con el fin de encontrar aquel entero mínimo necesario para lograr un valor de `out` tal que la diferencia entre `out` y 1 debe ser menor a 0.000001 (i.e.: `1e-06`). Puede comprobar que su resultado cumple con esa premisa con el comando `1 - out < 1e-06`.

Nota: en este ejercicio, y *a diferencia de la mayoría de los casos* de este curso, la respuesta para el valor de `n` no es general, si no que debe ser un número entero (i.e.: 42), el cual usted deberá escribir en la línea correspondiente del archivo "zenon.R".

Para visualizar la convergencia de su serie puede usar el comando:

```
plot(cumsum(s), type = "o", xlab = "n", ylab = expression(Z[n]))
```

### 2.c Extra: series geométricas

Script: geom.R

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

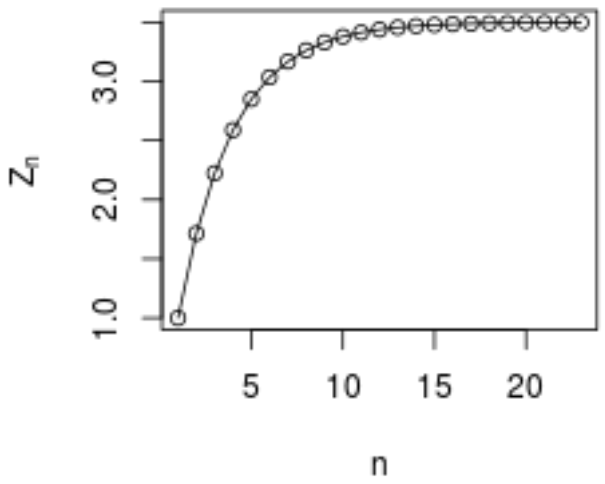


Figure 3: Serie de Zenón

La serie $Z_n$ es un caso particular de serie geométrica. La fórmula general de este tipo de series es:

$$S_n = \sum_{i=0}^{i=n} \frac{1}{z^i}$$

(Nota: ahora el primer término corresponde a i=0, por lo que a toda $S_n$ se le agrega un +1 en comparación con la misma sumatoria empezando por i=1.)

Un resultado importante es que la serie converge a un valor determinado (es decir, tiene un límite finito para $n \to \infty$) para todos los z tales que $1 < \|z\|$ (1 menor al valor absoluto de $z$). El objetivo de este ejercicio es confirmar este resultado. Para esto debe completarse el código del archivo `geom.R`, siguiendo las instrucciones que allí se indican.

Nota: en este ejercicio utilice valores para `n` y `z` a elección (siendo el primero entero y positivo).

Si el código es correcto, el resultado `out` debería ser idéntico (o casi igual, debido a pequeños errores numéricos de redondeo) al valor de la siguiente fórmula:

$$S_n = \frac{1 - z^{-(n-1)}}{1 - z^{-1}}$$

---

## 3. Índice de Shannon-Wiener

El índice de Shannon-Wiener o simplemente índice de Shannon ha sido utilizado en ecología y otras ciencias como indicador de diversidad de una comunidad de especies. Este índice fue creado originalmente para ser usado como medida de entropía en cadenas de caracteres (i.e., texto) en el contexto de la teoría de la información (Legendre & Legendre, 2012). En general para una colección de objetos de distintas categorías (i.e., individuos de distintas especies), el índice de Shannon se calcula con la siguiente fórmula:

$$H = -\sum_{i=1}^{i=S} p_i \cdot log_2(p_i)$$

(La base del logaritmo no es demasiado importante, pero en este ejercicio nos vamos a regir a esta definición en particular; es decir, vamos a usar logarítmo de base 2)

Aquí $S$ es el número total de categorías o especies y $p_i$ es la frecuencia relativa de la categoría $i$. Es decir:

$$p_i = \frac{n_i}{N}$$

En donde $n_i$ es la cantidad de objetos de la categoría $i$ contenidos en nuestra colección y $N$ es la cantidad total de objetos. Por ejemplo, para la palabra "banana", los $p_i$ de las letras "a", "b" y "n" son 3/6, 1/6 y 2/6 respectivamente y el $H$ resultante es 1.46.

### 3.a Cálculo de H para una colección de números

Script: shannon-1.R

En el archivo "shannon-1.R" usted encontrará código de R incompleto. El objetivo de este script es calcular el $H$ de un vector de números llamado `coleccion`. Pero a diferencia del ejercicio 2, vamos a utilizar otra estrategia para de hacer la sumatoria necesaria, a través del operador `%*%` de R.

Una forma de hacer sumatorias del tipo $\sum v_i \cdot u_i$ es usando el producto escalar o producto interior de vectores, para lo cual en R se usa el operador `%*%`. Específicamente, si $v$ y $u$ son vectores columna, entonces $\langle v, u \rangle = \sum v_i \cdot u_i$. Otra representación común es:

$$v^T \cdot u = (v_1 v_2 \cdots v_d) \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_d \end{pmatrix}$$

en dónde $v^T$ es el vector columna $v$ transpuesto y $d$ es la dimensión o número de elementos de ambos vectores. En R el producto escalar se escribe

```
v %*% u
```

simplemente, ignorando detalles de transposición. De hecho los vectores en R no son fila o columna, si no que simplemente son una secuencia de elementos. En general este operador sirve para toda multiplicación de matrices en R.

En este ejercicio deberá completar el archivo "shannon-1.R" de forma que sea capaz de calcular el valor de H para el vector `coleccion` o *cualquier otro vector* de R, utilizando el producto escalar de vectores para calcular el resultado de la sumatoria. Para el caso particular de `coleccion`, el $H$ esperado es de 2.699514.

```
coleccion <- c(9, 6, 3, 3, 6, 1, 5, 5, 5, 3, 2, 7, 2, 1)
```

El cálculo del $H$ puede dividirse en (1) calcular los valores de $p_i$ y (2) realizar la sumatoria. Para calcular los $p_i$ es necesario a su vez tener los $n_i$ y el valor $N$. Para estas tareas las funciones `table` y `length` pueden ser de gran ayuda. La primera sirve para hacer un conteo de la cantidad de objetos por categoría, mientras que la función `length` devuelve la cantidad de elementos de un vector cualquiera. No dude en consultar la documentación de R de estas dos funciones en caso de que no comprenda del todo bien su accionar.

Nota: para verificar que su solución sirve para cualquier vector, puede instalar el paquete `vegan`, el cual cuenta con la función `diversity` para calcular el índice Shannon-Wiener (aunque a partir de datos de conteo, a diferencia del vector `coleccion`). El siguiente código sirve como verificación:

```
install.packages("vegan")  # Si el paquete no está instalado
library(vegan)  # Para cargar el paquete
coleccion <- rpois(42, 6)  # Para crear un vector aleatorio
source("shannon-1.R")  # Calcula H
H1 <- diversity(table(coleccion), base = 2)  # Calcula H1
H - H1 < 1e-13  # Si es TRUE, su código está bien
```

### 3.b Extra: función `shannon`

Script: shannon-2.R

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

El archivo "shannon-2.R" contiene el código incompleto necesario para crear una función que calcule el índice de Shannon para cualquier vector de R. Si le resulta útil, utilice la función `prp` creada en la lección 1.2 ("una sesión de ejemplo") como referencia.

Tenga en cuenta que en los paréntesis vacíos de luego de `function` es en donde se ponen los argumentos o entradas de la función. Esta función sólo debería usar un argumento; el nombre elegido el mismo debe ser mismo que se use luego en los comandos internos de la misma (los comandos comprendidos entre las llaves {}).

Si su función está correcta, el siguiente código debería correr correctamente y el resultado debería ser el predicho por el objeto `frase`:

```
frase <- "Esta frase tiene un H = 3.7"
x <- strsplit(frase, "")[[1]]
shannon(x)
```

```
## [1] 3.708
```

(Nota: el segundo comando toma la `frase` y la parte en todos los caracteres que la componen; se podría también modificar el código de verificación del ejercicio anterior para utilizar la función `diversity` y compararla con la suya.)