Crash course de Bioinformática - Sesión 1

Aprender a programar en R es como aprender un lenguaje nuevo

En esta sesión vamos a practicar cómo programar funciones básicas que se usan mucho para analizar datos de secuenciación.

Qué hacer cuando no sabes qué hacer

RStudio tiene documentación muy extensa. Cada vez que no sepas cómo usar un comando, basta con escribir un *?* seguido del comando:

```
?matrix
```

Google es tu amigo!!!

Crea un notebook

Los notebooks son herramientas que facilitan la programación y te ayudan a que tu código se vea más limpio. 1. Da click en File. 2. En la opción de New File, selecciona "R Script".

Parte 1 - Fundamentos de R

Ya que abriste tu notebook en R, en la parte de abajo verás la consola, que es donde tus comandos van a correr. R tiene miles de "packages" de comandos que incrementan la funcionalidad. Por ejemplo:

```
library(MASS)
mean(x)
#Muchos de los "packages" contienen ejemplos y tutoriales de uso que se
#llaman "vignettes", que puedes acceder escribiendo:
browseVignettes()
```

En esta sesión no vamos a descargar *packages*.

Parte 2: Usando R como calculadora

Al igual que Matlab, R es un lenguaje matemático.

Intenta programar las siguientes ecuaciones básicas:

```
4+7
```

```
## [1] 11
```

```r
4-7
```

```
## [1] -3
```

```r
4*7
```

```
## [1] 28
```

```r
exp(3)
```

```
## [1] 20.08554
```

```r
sin(4)
```

```
## [1] -0.7568025
```

```r
log(0)
```

```
## [1] -Inf
```

```r
log(-1)  #--> NaN (Not a Number)
```

```
## Warning in log(-1): Se han producido NaNs
```

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

Parte 3: Vectores y Matrices

Los vectores y matrices nos van a ayudar a hacer tablas con nuestros datos, con lo que nos va a ser mucho más fácil trabajar.

```r
#Para crear un vector usamos el comando *c* (concatenate)
x=c(1,2,3,2,3)
#Esto crea un vector con 5 elementos que guardamos en la variable "x"
#¿Qué pasa si usamos diferentes símbolos para asignar números a variables?
y=c(1,2,3,2,3)
z<-c(1,2,3,2,3) #Puedes usar = o <- dependiendo de tus preferencias
```

Podemos hacer operaciones con los vectores. Trata de calcular:

x-1

x*2

x^2

exp(x)

Podemos indexar los vectores usando corchetes ([]). Para seleccionar un elemento (o un grupo de elementos) de un vector, escribe

```
x[5]
x[3:5]
x[c(3,5)]
x[c(1,4)]
#Ahora vamos a intentar esto; puedes explicar qué estamos haciendo?
a = 2:4
x[a]
x[x>2]
```

Para hacer matrices usamos el comando *matrix*, que necesita 3 factores: un vector (el relleno de la matriz), el número de filas (=) y el número de columnas (|||). Por ejemplo,

```
#Para hacer matrices usamos el comando *matrix*, que necesita 3 factores: un vector (el relleno de la m
z=matrix(1:12, 4,3)
#Para indexar matrices ahora usamos dos dimensiones en vez de una.
#Para hacer una matriz 2x2 basada en las filas 3 a 4 y en las columnas 2 a 3:
z[3:4,2:3]
```

El comando apply() es muy útil para hacer operaciones matemáticas con matrices:

```
#Usa ?apply para describir qué pasa cuando programamos la siguiente expresión
apply(z, 1, sum)
```

Usa *apply* y *mean* para calcular el valor promedio de cada *columna* en z.

Parte 4: Trabajando con Data Frames

Los Data Frames son la forma más versátil para organizar datos. Se ven iguales que las matrices, pero pueden contener más de un tipo de dato (como números y letras en vez de nomás números). En este ejercicio vamos a usar un dataset de ejemplo que viene incluido en RStudio.

```
library(MASS) #Con esta línea instalamos el paquete "MASS", que contiene los datos "mammals"
data(mammals) #Aquí cargamos el dataset "Mammals"
class(mammals) #Con esta línea podemos ver que mammals está cargado como data.frame
```

El data set mammals tiene 62 filas y 2 columnas. Podemos ver esto usando el comando *dim()*

Con el comando *head()* podemos ver las primeras 6 filas de nuestro data frame

```
head(mammals)
```

Podemos ver que los nombres de las filas y de las columnas no se cuentan en las dimensiones de los data frames. Los nombres se pueden extraer con

```
rownames(mammals)
colnames(mammals)
```

Para calcular el peso promedio de los mamíferos que tenemos en el dataframe “mammals”, podemos escribir

```
mean(mammals[,1])
#Ahora para el peso promedio de su cerebro,
mean(mammals[,2])
```

> Ahora, usa *median*, *var* y *sd* para calcular la mediana, varianza y desviación estándar del peso del cuerpo y del cerebro. Ten en cuenta que el peso del cuerpo está en kg y el cerebro en g (de acuerdo a la documentación en ?mammals).

Ahora, nuestro objetivo es saber qué mamífero tiene el cerebro más grande en comparación con su cuerpo:

```
ratio = (mammals[,2]/1000)/mammals[,1]
```

Esta información se puede añadir fácilmente al dataframe de mammals con

```
mammals2=cbind(mammals, ratio)
```

El nombre de la nueva columna se va a poner automáticamente como "ratio", pero podemos cambiar eso con

```
colnames(mammals2)=c("Body", "Brain", "Body/Brain-ratio")
```

Es hora de ordenar nuestra tabla, y podemos hacerlo con la función *order()*

```
mammals.order=order(mammals2[,3], decreasing=TRUE)
#Decreasing=TRUE nos va a ordenar los datos de más alto a más pequeño.
#Para hacerlo al revés, usa decreasing=FALSE.
#Aún no terminamos. Ahora que tenemos el nuevo orden, indexamos mammals2 para
#que se organice como queremos
mammals2.sorted=mammals2[mammals.order,]
#la coma después de mammals.order
#es muy importante para decirle al data frame que debe de ordenar sus filas y
#no sus columnas de esa forma.
```

> ¿Qué mamífero tiene la relación Body/Brain más alta? ¿Cuál tiene la más baja? (Tip: usa los comandos head() y tail()).

##Parte 5: Usando funciones estadísticas Ahora vamos a usar otro dataset:

```
data(cats)
#Qué dimensiones tiene este data set?
```

Para usar la información de gatos hembras, escribimos:

```
female = cats[,"Sex"] == "F"
```

El operador lógico == nos regresa una matriz lógica de "FALSE"s y "TRUE"s que luego podemos indexar para hacer un nuevo data frame sólo con los datos de los gatos hembra:

```
cats.female = cats[female,]
#El comando summary() nos da un análisis rápido de los datos:
summary(cats.female)
```

Examina el data frame *cats.female* y asegúrate de que sólo contenga datos de los gatos hembra. Luego, repite esta operación con los gatos macho. Guarda los resultados en la variable *"cats.male"*. Con *summary()* calcula la *mediana*, *media*, *varianza* y *desviación estándar* de ambos data sets. Son similares?

Como discutimos antes, R es muy útil para analizar diferencias estadísticas. F-test

```
?var.test
var.test(cats.female[,2], cats.male[,2])
```

Interpreta los resultados. La diferencia es significativa?

Parte 6: Loops

Los loops nos pueden ayudar cuando tenemos que hacer una acción repetitiva muchas veces. Por ejemplo, si queremos calcular la suma de todos los números enteros de 5 a 514, podemos elaborar el siguiente loop:

```
num = 5 #primero establecemos un vector o matriz con el que vamos a trabajar
for (i in 6:514){ #Establecemos una variable que se va a sumar con cada iteración
  num = num + i #En cada iteración, "num" se reemplaza
  }
print(num) #Después de que las ~500 iteraciones se acabaron, imprimimos el último
#valor de "num". Si esta línea hubiera estado adentro del loop, hubiéramos
#impreso todos los ~500 valores que "num" tuvo.
```

Usando un loop, intenta calcular la suma de $1 + 1/2 + 1/3 + \ldots + 1/10$

```
l = 0
for (j in 1:10){
  l = l + 1/j
}
print(l)
```

```
## [1] 2.928968
```

Parte 7: Graficando y visualizando los datos

```
plot(cats[,2], cats[,3])
```

Usando ?plot, construye una gráfica con cats[,2] y cats[,3] en la que los puntos graficados sean triángulos rojos, con un título que diga "Cat body-heart weight", que el eje x tenga el título "Body weight" y el eje y tenga el título "Heart weight"

Terminamos esta sesión!!!

Recuerda que al igual que con cualquier lenguaje, la clave para aprender a programar es la práctica. Aún si no pudiste terminar este ejercicio hoy, intenta terminarlo en casa (: