



# Introducción a Data Science: Programación Estadística con R

Diego Pérez Rojas


En este informe se muestra evidencia de realización del curso online de Coursera, incluyendo las secciones requeridas por el programa para poder completar el curso de manera satisfactoria y cumpliendo con el 100% completado a su totalidad.

diealpero/talleresinferen x HTML5 UPI Responsive x Madura (feat. Bad Bunni) x Introducción a Data Sci x swirldev/swirl\_courses / x Aqueronte: R: Tipos de x

www.coursera.org/learn/intro-data-science-programacion-estadistica-r/home/welcome

Amazon.com Instagram Videos de los Viernes MaxiHP - Video: Drift Takora® Online : Veri Uhp Online - Ultra H Free Ebooks and Com Mapas SII Estas 100 frases son

**coursera** Explorar What do you want to learn?



**Página de inicio del curso**

- Semana 1
- Semana 2
- Semana 3
- Semana 4
- Calificaciones
- Foros de debate
- Mensajes
- Información del Curso

**¡Felicitaciones!**

Completaste correctamente **Introducción a Data Science: Programación Estadística con R**

Comprar el Curso

★★★★★

✓ SEMANA 1

✓ SEMANA 2

✓ SEMANA 3

✓ SEMANA 4

Si, quiero recibir correos electrónicos sobre otros programas de **Universidad Nacional Autónoma de México**.


Si

diealpero/talleresinferen x HTML5 UPI Responsive x Madura (feat. Bad Bunni) x Introducción a Data Sci x swirldev/swirl\_courses / x Aqueronte: R: Tipos de x

www.coursera.org/learn/intro-data-science-programacion-estadistica-r/home/assignments

Amazon.com Instagram Videos de los Viernes MaxiHP - Video: Drift Takora® Online : Veri Uhp Online - Ultra H Free Ebooks and Com Mapas SII Estas 100 frases son

**coursera** Explorar What do you want to learn?



**Página de inicio del curso**

**Calificaciones**

- Foros de debate
- Mensajes
- Información del Curso

**100%**

Calificación del Curso

	Vencimiento	Peso	Aprobado	Calificación
<b>Introducción al Lenguaje</b>				
✓ <b>Tareas de programación:</b> Lección de swirl: 1. Obtener Ayuda 3h	Jun 24	2%	✓	100.00%
✓ <b>Tareas de programación:</b> Lección de swirl: 2. Objetos Tipos de Datos y Operaciones Básicas 3h	Jun 24	10%	✓	100.00%
✓ <b>Tareas de programación:</b> Lección de swirl: 3. Subconjuntos de Datos 3h	Jun 24	10%	✓	100.00%
✓ <b>Tareas de programación:</b> Lección de swirl: 4. Leer y Escribir Datos 3h	Jun 24	10%	✓	100.00%

The image displays two screenshots of the Coursera course 'Introducción a Data Science' (Introduction to Data Science) showing the progress of assignments. The course is part of the 'swirdev/swirl\_courses' track.

**Top Screenshot: Utilización del Lenguaje**

Asignación	Fecha	Progreso	Estado	Puntuación
Tareas de programación: Lección de swirl: 5. Funciones	Jul 1	10%	✓	100.00%

**Bottom Screenshot: Acercamiento al Sistema de Gráficos de R**

Asignación	Fecha	Progreso	Estado	Puntuación
Tareas de programación: Lección de swirl: 6. Funciones apply	Jul 8	15%	✓	100.00%
Tareas de programación: Lección de swirl: 7. Graficación	Jul 8	15%	✓	100.00%
Tareas de programación: Lección de swirl: 8. Parámetros en el Sistema de Gráficos	Jul 8	5%	✓	100.00%
Tareas de programación: Lección de swirl: 9. Colores en el Sistema de Gráficos	Jul 8	5%	✓	100.00%

**Bottom Screenshot: Expresiones Regulares, Graficación con ggplot2 y Simulación**

Asignación	Fecha	Progreso	Estado	Puntuación
Tareas de programación: Lección de swirl: 12. Expresiones Regulares	Jul 15	8%	✓	100.00%
Tareas de programación: Lección de swirl: 14. Simulación	Jul 15	10%	✓	100.00%

A continuación se muestran los códigos de todas las secciones realizadas

## Sección 1

- 1: Obtener Ayuda
- 2: Objetos Tipos de Datos y Operaciones Basicas
- 3: Subconjuntos de Datos
- 4: Leer y escribir Datos
- 5: Funciones
- 6: Funciones apply
- 7: Graficacion
- 8: Parametros en el Sistema de Graficos
- 9: Colores en el Sistema de Graficos
- 10: Graficacion con texto y notacion matematica
- 11: Creacion de Graficas en 3D
- 12: Expresiones regulares
- 13: Graficacion con ggplot2
- 14: Simulacion

selection: 1

| 0%

| En esta lección conocerás las principales herramientas que  
| R tiene para obtener ayuda.

...

|===| 6%

| La primera herramienta que puedes usar para obtener ayuda  
| es `help.start()`. En ella encontrarás un menú de recursos,  
| entre los cuales se encuentran manuales, referencias y  
| demás material para comenzar a aprender R.

...

|=====| 12%

| Para usar `help.start()` escribe en la línea de comandos  
| `help.start()`. Pruébalo ahora:

> `help.start()`

If nothing happens, you should open  
'<http://127.0.0.1:25292/doc/html/index.html>' yourself

| Perseverancia es la respuesta.

|=====| 18%

| R incluye un sistema de ayuda que te facilita obtener  
| información acerca de las funciones de los paquetes  
| instalados. Para obtener información acerca de una función,  
| por ejemplo de la función `print()`, debes escribir `?print` en  
| la línea de comandos.

...

|=====| 24%

| Ahora es tu turno, introduce `?print` en la línea de  
| comandos.

```
> ?print
```

```
| ¡Es asombroso!
```

```
|=====| 29%  
| Como puedes observar ? te muestra en la ventana Help una  
| breve descripción de la función, de cómo usarla, así como  
| sus argumentos, etcétera.
```

```
...
```

```
|=====| 35%  
| Asimismo, puedes usar la función help(), la cual es un  
| equivalente de ?. Al utilizar help(), usarás como argumento  
| el nombre de la función entre comillas, por ejemplo,  
| help("print").
```

```
...
```

```
|=====| 41%  
| Para buscar ayuda sobre un operador, éste tiene que  
| encontrarse entre comillas inversas. Por ejemplo, si buscas  
| información del operador +, deberás escribir help(`+`) o  
| ?`+` en la línea de comandos.
```

```
...
```

```
|=====| 47%  
| Otra herramienta disponible es la función apropos(), la  
| cual recibe una cadena entre comillas como argumento y te  
| muestra una lista de todas las funciones que contengan esa  
| cadena. Inténtalo: escribe apropos("class") en la línea de  
| comandos.
```

```
> apropos("class")
```

```
[1] ".checkMFCclasses"  
[2] ".classEnv"  
[3] ".MFclass"  
[4] ".OldClassesList"  
[5] ".rs.getR6ClassGeneratorMethod"  
[6] ".rs.getR6ClassSymbols"  
[7] ".rs.getSetRefClassSymbols"  
[8] ".rs.getSingleClass"  
[9] ".rs.objectClass"  
[10] ".rs.rnb.engineToCodeClass"  
[11] ".rs.rpc.get_set_class_slots"  
[12] ".rs.rpc.get_set_ref_class_call"  
[13] ".selectSuperClasses"  
[14] ".valueClassTest"  
[15] "all.equal.envRefClass"  
[16] "assignClassDef"  
[17] "class"  
[18] "class<-"  
[19] "classesToAM"  
[20] "classLabel"  
[21] "classMetaName"  
[22] "className"
```

```

[23] "completeClassDefinition"
[24] "completeSubclasses"
[25] "data.class"
[26] "findClass"
[27] "getAllSuperClasses"
[28] "getClass"
[29] "getClassDef"
[30] "getClasses"
[31] "getClassName"
[32] "getClassPackage"
[33] "getRefClass"
[34] "getSubclasses"
[35] "insertClassMethods"
[36] "isClass"
[37] "isClassDef"
[38] "isClassUnion"
[39] "isSealedClass"
[40] "isVirtualClass"
[41] "isXS3Class"
[42] "makeClassRepresentation"
[43] "makePrototypeFromClassDef"
[44] "multipleClasses"
[45] "namespaceImportClasses"
[46] "nclass.FD"
[47] "nclass.scott"
[48] "nclass.Sturges"
[49] "newClassRepresentation"
[50] "oldClass"
[51] "oldClass<-"
[52] "promptClass"
[53] "removeClass"
[54] "resetClass"
[55] "S3Class"
[56] "S3Class<-"
[57] "sealClass"
[58] "selectSuperClasses"
[59] "setClass"
[60] "setClassUnion"
[61] "setOldClass"
[62] "setRefClass"
[63] "showClass"
[64] "superClassDepth"
[65] "unclass"

```

| ¡Buen trabajo!

```

|=====| 53%
| También puedes obtener ejemplos del uso de funciones con la
| función example(). Por ejemplo, escribe
| example("read.table").

```

```
> example("read.table")
```

```

rd.tbl> ## using count.fields to handle unknown maximum number of fields
rd.tbl> ## when fill = TRUE
rd.tbl> test1 <- c(1:5, "6,7", "8,9,10")

```

```

rd.tbl> tf <- tempfile()

rd.tbl> writeLines(test1, tf)

rd.tbl> read.csv(tf, fill = TRUE) # 1 column
  x1
1  2
2  3
3  4
4  5
5  6
6  7
7  8
8  9
9 10

rd.tbl> ncol <- max(count.fields(tf, sep = ","))

rd.tbl> read.csv(tf, fill = TRUE, header = FALSE,
rd.tbl+   col.names = paste0("v", seq_len(ncol)))
  v1 v2 v3
1  1 NA NA
2  2 NA NA
3  3 NA NA
4  4 NA NA
5  5 NA NA
6  6  7 NA
7  8  9 10

rd.tbl> unlink(tf)

rd.tbl> ## "Inline" data set, using text=
rd.tbl> ## Notice that leading and trailing empty lines are auto-trimmed
rd.tbl>
rd.tbl> read.table(header = TRUE, text = "
rd.tbl+ a b
rd.tbl+ 1 2
rd.tbl+ 3 4
rd.tbl+ ")
  a b
1 1 2
2 3 4

```

| ¡Buen trabajo!

```

|=====| 59%
| Con eso tendrás una idea de lo que puedes hacer con esta
| función.

```

...

```

|=====| 65%
| R te permite buscar información sobre un tema usando ??.
| Por ejemplo, escribe ??regression en la línea de comandos.

```

> ??regression



| ¡Todo ese trabajo está rindiendo frutos!

|=====| 71%  
| Esta herramienta es muy útil si no recuerdas el nombre de  
| una función, ya que R te mostrará una lista de temas  
| relevantes en la venta Help. Análogamente, puedes usar la  
| función `help.search("regression")`.

...

|=====| 76%  
| Otra manera de obtener información de ayuda sobre un  
| paquete es usar la opción `help` para el comando `library`, con  
| lo cual tendrás información más completa. Un ejemplo es  
| `library(help="stats")`.

...

|=====| 82%  
| Algunos paquetes incluyen viñetas. Una viñeta es un  
| documento corto que describe cómo se usa un paquete. Puedes  
| ver una viñetas usando la función `vignette()`. Pruébalo:  
| escribe `vignette("tests")` en la línea de comandos.

> `vignette("tests")`

| ¡Eso es correcto!

|=====| 88%  
| Por último si deseas ver la lista de viñetas disponibles  
| puedes hacerlo usando el comando `vignette()` con los  
| paréntesis vacíos.

...

|=====| 94%  
| Es MUY IMPORTANTE que sepas que durante todo el curso en `swirl`, puedes  
| hacer uso de las funciones `help()` o `?` cuando lo  
| desees, incluso si estas en medio de una lección.

...

|=====| 100%  
| Has concluido la lección. ¿Te gustaría que se le notificará a Coursera  
| que has completado esta lección?

## Sección 2

| Selecciona una lección por favor, o teclea 0 para volver al menú del cu  
| rso.

1: Obtener Ayuda  
tos y Operaciones Basicas

2: Objetos Tipos de Da

3: Subconjuntos de Datos	4: Leer y escribir Datos
5: Funciones	6: Funciones apply
7: Graficacion	8: Parametros en el Si
9: Colores en el Sistema de Graficos	10: Graficacion con texto y notacion matematica
11: Creacion de Graficas en 3D	12: Expresiones regulares
13: Graficacion con ggplot2	14: Simulacion

## Selection: 2

```
|
| 0%
```

| En esta lección conocerás los tipos de datos que existen en el lenguaje R, además de las operaciones básicas que puedes hacer con ellos.

...

```
|=
| 1%
```

| Cuando introduces una expresión en la línea de comandos y das ENTER, R evalúa la expresión y muestra el resultado (si es que existe uno). R puede ser usado como una calculadora, ya que realiza operaciones aritméticas, además de operaciones lógicas.

...

```
|===
| 2%
```

| Pruébalo: ingresa 3 + 7 en la línea de comandos.

```
> 3+7
[1] 10
```

| ¡Tu dedicación es inspiradora!

```
|====
| 3%
```

| R simplemente imprime el resultado 10 por defecto. Sin embargo, R es un lenguaje de programación y normalmente la razón por la que usas éstos es para automatizar algún proceso y evitar la repetición innecesaria.

...

```
|=====
| 4%
```

| En ese caso, tal vez quieras usar el resultado anterior en algún otro cálculo. Así que en lugar de volver a teclear la expresión cada vez que la necesites, puedes crear una variable que guarde el resultado de ésta.

...

|=====

| 6%

| La manera de asignar un valor a una variable en R es usar el operador de asignación, el cual es sólo un símbolo de menor que seguido de un signo de menos, mejor conocido como guion alto. El operador se ve así: <-

...

|=====

| 7%

| Por ejemplo, ahora ingresa en la línea de comandos: `mi_variable <- (180 / 6) - 15`

```
> mi_variable <- (180 / 6) - 15
```

| ¡Buen trabajo!

|=====

| 8%

| Lo que estás haciendo en este caso es asignarle a la variable `mi_variable` el valor de todo lo que se encuentra del lado derecho del operador de asignación, en este caso `(180 / 6) - 15`.

...

|=====

| 9%

| En R también puedes asignar del lado izquierdo: `(180 / 6) - 15 -> mi_variable`

...

|=====

| 10%

| Como ya te habrás dado cuenta, la asignación '`<-`' no muestra ningún resultado. Antes de ver el contenido de la variable '`mi_variable`', ¿qué crees que contenga la variable '`mi_variable`'?

- 1: la expresión `(180 / 6) - 15`
- 2: la dirección de memoria de la variable '`mi_variable`'
- 3: la expresión evaluada, es decir un 15

Selection: 3

| ¡Excelente trabajo!

|=====

| 11%

| La variable '`mi_variable`' deberá contener el número 15, debido a que  $(180 / 6) - 15 = 15$ . Para revisar el contenido de una variable, basta con escribir el nombre de ésta en la línea de comandos y presionar ENTER. Inténtalo: muestra el contenido de la variable '`mi_variable`':

```

> mi_variable
[1] 15

| Perseverancia es la respuesta.

| =====
| 12%
| Nota que el '[1]' acompaña a los valores mostrados al evaluar las expresiones anteriores. Esto se debe a que en R todo
| número que introduces en la consola es interpretado como un vector.

...

| =====
| 13%
| Un vector es una colección ordenada de números, por lo cual el '[1]' de
| nota la posición del primer elemento mostrado en
| el renglón 1. En los casos anteriores sólo existe un único elemento en
| el vector.

...

| =====
| 15%
| En R puedes construir vectores más largos usando la función c() (combin
| e). Por ejemplo, introduce: y <- c(561, 1105,
| 1729, 2465, 2821)

> y <- c(561, 1105, 1729, 2465, 2821)

| ¡Buen trabajo!

| =====
| 16%
| Ahora observa el contenido de la variable 'y'. Otra manera de ver el co
| ntenido de una variable es imprimirlo con la
| función print(). Introduce print(y) en la línea de comandos:

> print(y)
[1] 561 1105 1729 2465 2821

| ¡Excelente!

| =====
| 17%
| Como puedes notar, la expresión anterior resulta ser un vector que cont
| iene los primeros cinco números de Carmichael.
| Como ejemplo de un vector que abarque más de una línea, usa el operador
| de secuencia para producir un vector con cada
| uno de los enteros del 1 al 100. Introduce 1:100 en la línea de comando
| s.

> 1:100
[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
18 19 20 21 22 23 24 25 26 27 28 29
[30] 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46
47 48 49 50 51 52 53 54 55 56 57 58

```

```
[59] 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75
76 77 78 79 80 81 82 83 84 85 86 87
[88] 88 89 90 91 92 93 94 95 96 97 98 99 100
```

| ¡Muy bien!

```
|=====
| 18%
| El vector es el objeto más simple en R. La mayoría de las operaciones e
stán basadas en vectores.
```

...

```
|=====
| 19%
| Por ejemplo, puedes realizar operaciones sobre vectores y R automáticam
ente empareja los elementos de los dos vectores.
| Introduce c(1.1, 2.2, 3.3, 4.4) - c(1, 1, 1, 1) en la línea de comandos
.
```

```
> c(1.1, 2.2, 3.3, 4.4) - c(1, 1, 1, 1)
[1] 0.1 1.2 2.3 3.4
```

| Esa es la respuesta que estaba buscando.

```
|=====
| 20%
| Nota: Si los dos vectores son de diferente tamaño, R repetirá la secuen
cia más pequeña múltiples veces. Por ejemplo,
| introduce c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10) - c(1, 2) en la línea de com
andos.
```

```
> c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10) - c(1, 2)
[1] 0 0 2 2 4 4 6 6 8 8
```

| ¡Eres el mejor!

```
|=====
| 21%
| En R casi todo es un objeto. Para ver qué objetos tienes en un momento
determinado, puedes usar la función ls().
| Inténtalo ahora.
```

```
> ls()
[1] "mi_variable" "ncol"          "test1"         "tf"            "y"
```

| ¡Buen trabajo!

```
|=====
| 22%
| Como sabes, existen otros tipos de objetos, como los caracteres (charac
ter).
```

...

```
|=====
| 24%
```

| Las expresiones con caracteres se denotan entre comillas. Por ejemplo, introduce "¡Hola Mundo!" en la línea de comandos.

```
> "¡Hola Mundo!"  
[1] "¡Hola Mundo!"
```

| ¡Eso es trabajo bien hecho!

|=====

| 25%

| Esto es mejor conocido en R como un vector de caracteres. De hecho, este ejemplo es un vector de longitud uno.

...

|=====

| 26%

| Ahora crea una variable llamada 'colores' que contenga un vector con las cadenas "rojo", "azul", "verde", "azul", "rojo", en ese orden.

```
> colores<- c("rojo", "azul", "verde", "azul","rojo")
```

| ¡Lo estás haciendo muy bien!

|=====

| 27%

| Ahora imprime el vector 'colores' .

```
> colores  
[1] "rojo" "azul" "verde" "azul" "rojo"
```

| ¡Mantén este buen nivel!

|=====

| 28%

| En otros lenguajes como C, carácter (character) hace referencia a un simple carácter, y cadena (string) se entiende como un conjunto de caracteres ordenados. Una cadena de caracteres es equivalente al valor de carácter en R.

...

|=====

| 29%

| Además, hay objetos de tipo numérico (numeric) que se dividen en complejos (complex) y enteros (integer). Los últimos ya los conoces, pues has estado trabajando con ellos, además de los vectores y los caracteres.

...

|=====

| 30%

| Los complejos en R se representan de la siguiente manera: a+bi, donde 'a' es la parte real y 'b' la parte imaginaria.

| Pruébalo: guarda el valor de 2+1i en la variable 'complejo'.

```
> complejo<- 2+1i
```

| ¡Traes una muy buena racha!

```
|=====
| 31%
| Al igual que los demás objetos de tipo numérico, lo complejos pueden ha
cer uso de los operadores aritméticos más
| comunes, como `+` (suma), `-` (resta, o negación en el caso unario), `/`
(división), `*` (multiplicación) `^` (donde x^2
| significa 'x elevada a la potencia 2'). Para obtener la raíz cuadrada,
usa la función sqrt(), y para obtener el valor
| absoluto, la función abs().
```

...

```
|=====
| 33%
| También hay objetos lógicos (logic) que representan los valores lógicos
falso y verdadero.
```

...

```
|=====
| 34%
| El valor lógico falso puede ser representado por la instrucción FALSE o
únicamente por la letra F mayúscula; de la
| misma manera, el valor lógico verdadero es representado por la instrucc
ión TRUE o por la letra T.
```

...

```
|=====
| 35%
| Como operadores lógicos están el AND lógico: `&` y `&&` y el OR lógico:
`|` y `||`.
```

...

```
|=====
| 36%
| También existen operadores que devuelven valores lógicos, éstos pueden
ser de orden, como: `>` (mayor que), `<` (menor
| que), `>=` (mayor igual) y `<=` (menor igual), o de comparación, como:
`==` (igualdad) y `!=` (diferencia). Por
| ejemplo, introduce en la línea de comandos mi_variable == 15.
```

```
> mi_variable == 15
```

```
[1] TRUE
```

| Perseverancia es la respuesta.

```
|=====
| 37%
```

| Como puedes ver, R te devuelve el valor TRUE, pues si recuerdas, en la variable 'mi\_variable' asignaste el valor de la expresión  $(180 / 6) - 15$ , la cual resultaba en el valor 15. Por lo cual, cuando le preguntas a R si 'mi\_variable' es igual a 15, te devuelve el valor TRUE.

...

|=====

| 38%

| En R existen algunos valores especiales.

...

|=====

| 39%

| Por ejemplo, los valores NA son usados para representar valores faltantes. Supón que cambias el tamaño de un vector a un valor más grande del previamente definido. Recuerda el vector 'complejo', el cual contenía el número complejo 2+1i; cambia la longitud de 'complejo'. Ingresa `length(complejo) <- 3` en la línea de comandos.

> `length(complejo) <- 3`

| ¡Excelente!

|=====

| 40%

| Ahora ve el contenido de 'complejo'.

>

> `complejo`

[1] 2+1i NA NA

| ¡Bien hecho!

|=====

| 42%

| Los nuevos espacios tendrán el valor NA, el cual quiere decir not available (no disponible).

...

|=====

| 43%

| Si un resultado de la evaluación de alguna expresión aritmética es muy grande, R regresa el valor 'Inf' para un valor positivo y '-Inf' para un valor negativo (infinitos positivo y negativo, respectivamente). Por ejemplo, introduce `2^1024` en la línea de comandos.

> `2^1024`

[1] Inf

| ¡Eso es trabajo bien hecho!



```
|=====
| 44%
| Algunas veces la evaluación de alguna expresión no tendrá sentido. En e
stos casos, R regresará el valor Nan (not a
| number). Por ejemplo, divide 0 entre 0.
```

```
> 0/0
[1] NaN
```

```
| ¡Traes una muy buena racha!
```

```
|=====
| 45%
| Adicionalmente, en R existe el objeto null y es representado por el sím
bolo NULL.
```

```
...
```

```
|=====
| 46%
| Nota que NULL no es lo mismo que NA, Inf, -Inf o Nan.
```

```
...
```

```
|=====
| 47%
| Recuerda que R incluye un conjunto de clases para representar fechas y
horas. Algunas de ellas son: Date, POSIXct y
| POSIXlt.
```

```
...
```

```
|=====
| 48%
| Por ejemplo, introduce fecha_primer_curso_R <- Sys.Date() en la línea d
e comandos.
```

```
> fecha_primer_curso_R <- Sys.Date()
```

```
| ¡Eso es correcto!
```

```
|=====
| 49%
| Ahora imprime el contenido de fecha_primer_curso_R.
```

```
> fecha_primer_curso_R
[1] "2018-06-27"
```

```
| ¡Sigue trabajando de esa manera y llegarás lejos!
```

```
|=====
| 51%
| Recuerda que R te permite llevar a cabo operaciones numéricas y estadís
ticas con las fechas y horas. Además, R incluye
| funciones para manipularlas. Muchas funciones de graficación requieren
fechas y horas.
```

...

```
|=====
| 52%
| Ahora que conoces los objetos más usados en R, debes saber que además de la función c(), en R existe la función
| vector() para crear vectores. Al usar la función vector(), debes especificar el tipo de dato que almacenará el vector y
| el tamaño.
```

...

```
|=====
| 53%
| La función vector() crea un vector con los valores por defecto del tipo especificado.
```

...

```
|=====
| 54%
| Por ejemplo, escribe vector("numeric", length = 10) en la línea de comandos.
```

```
> vector("numeric", length = 10)
[1] 0 0 0 0 0 0 0 0 0 0
```

```
| ¡Buen trabajo!
```

```
|=====
| 55%
| Como puedes observar, el vector fue llenado con ceros.
```

...

```
|=====
| 56%
| Ahora introduce vector("character", length = 10) en la línea de comandos.
```

```
> vector("character", length = 10)
[1] "" "" "" "" "" "" "" "" "" ""
```

```
| ¡Muy bien!
```

```
|=====
| 57%
| Esta vez el vector fue llenado con 10 cadenas vacías.
```

...

```
|=====
| 58%
| Si le indicaras al vector que el tipo fuera "logical", ¿qué crees que contendría?
```

```
1: sólo valores FALSE
```

2: sólo valores TRUE  
3: ninguna de las anteriores

Selection: 2

| No exactamente. Dele otra oportunidad.

1: ninguna de las anteriores  
2: sólo valores FALSE  
3: sólo valores TRUE

Selection: 3

| Intenta de nuevo. ¡De cualquier forma hacerlo bien a la primera es aburrido!

1: sólo valores FALSE  
2: ninguna de las anteriores  
3: sólo valores TRUE

Selection:

Enter an item from the menu, or 0 to exit

Selection: 1

| ¡Mantén este buen nivel!

|=====

| 60%

| Crea un vector de tipo "logical" de tamaño 10 usando la función vector(  
| ).

> vector("logical",length=10)

[1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE

| ¡Sigue trabajando de esa manera y llegarás lejos!

|=====

| 61%

| Nota: Ahora que conoces los objetos más usados en R, es importante que  
| sepas que los vectores sólo pueden contener  
| objetos de la misma clase. Para guardar objetos de diferentes clases, p  
| uedes usar listas.

...

|=====

| 62%

| Las listas son una colección ordenada de objetos. Para crear una lista,  
| usa la función list() y especifica el contenido  
| de la lista separado por comas dentro de los paréntesis. Inténtalo: cre  
| a una lista que contenga un 0, la cadena "Hola"  
| y el valor TRUE, en ese orden.

> [0,"hola",TRUE]

Error: unexpected '[' in "["

```
> list(0,"hola",TRUE)
[[1]]
[1] 0

[[2]]
[1] "hola"

[[3]]
[1] TRUE
```

| Intenta de nuevo. ¡De cualquier forma hacerlo bien a la primera es aburrido! O escribe info() para más opciones.

| Introduce list(0,"Hola",TRUE) en la línea de comandos.

```
> list(0,"Hola",TRUE)
[[1]]
[1] 0

[[2]]
[1] "Hola"

[[3]]
[1] TRUE
```

| ¡Acertaste!

```
|=====
| 63%
| Anteriormente viste que en R los vectores sólo pueden contener objetos
de la misma clase.
```

...

```
|=====
==| 64%
| Pero, ¿qué pasa si creas un vector c(T, 19, 1+3i)? Introduce c(T, 19, 1
+3i) en la línea de comandos.
```

```
> c(T, 19, 1+3i)
[1] 1+0i 19+0i 1+3i
```

| ¡Lo has logrado! ¡Buen trabajo!

```
|=====
==| 65%
| Como habrás supuesto, el número complejo 1+3i no puede ser convertido a
entero ni a objeto de tipo "logical", entonces
| los valores T y 19 son convertidos a los números complejos 1+0i y 19+0i
respectivamente. Esto no es más que la
| representación de esos valores en objeto tipo "complex".
```

...

```
|=====
====| 66%
| Esto se llama coerción.
```

...

```
|=====
====| 67%
| La coerción hace que todos los objetos de un vector sean de una misma clase. Entonces, cuando creas un vector de
| diferentes tipos, R busca un tipo común, y los elementos que no son de
| ese tipo son convertidos.
```

...

```
|=====
====| 69%
| Otro ejemplo de coerción es cuando usas las funciones as.*().
```

...

```
|=====
====| 70%
| Inténtalo: crea un vector de longitud 5 de tipo "numeric" con la función vector() y guardarlo en la variable 'c'.
```

```
> c<-vector("numeric", length = 5)
```

| ¡Mantén este buen nivel!

```
|=====
====| 71%
| Revisa el contenido de la variable 'c' .
```

```
> c
[1] 0 0 0 0 0
```

| Perseverancia es la respuesta.

```
|=====
====| 72%
| Ahora usa la función as.logical() con el vector c.
```

```
> as.logical()
logical(0)
```

| ¡Sigue intentando! O escribe info() para más opciones.

| Introduce as.logical(c) en la línea de comandos.

```
> as.logical(c)
[1] FALSE FALSE FALSE FALSE FALSE
```

| ¡Mantén este buen nivel!

```
|=====
====| 73%
```

| Como puedes imaginar, el vector de tipo "numeric" fue explícitamente convertido a "logical".

...

|=====| 74%  
| Este tipo de coerción es mejor conocida como coerción explícita. Además de as.logical(), también existe as.numeric(), as.character(), as.integer().

...

|=====| 75%  
| Si usas la función class(), que te dice la clase a la que pertenece un objeto, obtendrás que class(c) = "numeric."  
| Pruébalo, ingresa class(c) en la línea de comandos.

```
> class(c)
[1] "numeric"
```

| ¡Tu dedicación es inspiradora!

|=====| 76%  
| Pero si después pruebas la misma función class() enviándole como argumento as.logical(c), obtendrás que es de tipo logical. Compruébalo:

```
> as.logical(c)
[1] FALSE FALSE FALSE FALSE FALSE
```

| No exactamente. Dele otra oportunidad. O escribe info() para más opciones.

| Introduce class(as.logical(c)) en la línea de comandos.

```
> class(as.logical(c))
[1] "logical"
```

| ¡Mantén este buen nivel!

|=====| 78%  
| Además de los vectores y las listas, existen las matrices.

...

|=====| 79%  
| Una matriz es una extensión de un vector de dos dimensiones. Las matrices son usadas para representar información de un solo tipo de dos dimensiones.

...

```
|=====
=====| 80%
| Una manera de generar una matriz es al usar la función matrix(). Inténtalo, introduce m <-
| matrix(data=1:12,nrow=4,ncol=3) en la línea de comandos.
```

```
>
> m<- matrix(data=1:12,nrow=4,ncol=3)
```

```
| ¡Tu dedicación es inspiradora!
```

```
|=====
=====| 81%
| Ahora imprime el contenido de 'm'.
```

```
> m
      [,1] [,2] [,3]
[1,]     1     5     9
[2,]     2     6    10
[3,]     3     7    11
[4,]     4     8    12
```

```
| ¡Toda esa práctica está rindiendo frutos!
```

```
|=====
=====| 82%
| Como puedes observar, creaste una matriz con tres columnas (ncol) y cuatro renglones (nrow).
```

```
...
```

```
|=====
=====| 83%
| Recuerda que también puedes crear matrices con las funciones cbind, rbind y as.matrix().
```

```
...
```

```
|=====
=====| 84%
| Los factores son otro tipo especial de vectores usados para representar datos categóricos, éstos pueden ser ordenados o no.
```

```
...
```

```
|=====
=====| 85%
| Recuerda el vector 'colores' que creaste previamente y supón que representa un conjunto de observaciones acerca de cuál es el color preferido de las personas.
```

```
...
```

```
|=====
=====| 87%
```

| Es una representación perfectamente válida, pero puede llegar a ser ineficiente. Ahora representarás los colores como un factor. Introduce `factor(colores)` en la línea de comandos.

```
>  
> factor(colores)  
[1] rojo azul verde azul rojo  
Levels: azul rojo verde
```

| ¡Mantén este buen nivel!

```
|=====| 88%  
| La impresión de un factor muestra información ligeramente diferente a la de un vector de caracteres. En particular, puedes notar que las comillas no son mostradas y que los niveles son explícitamente impresos.
```

...

```
|=====| 89%  
| Por último, existen los dataframes, que son una manera muy útil de representar datos tabulares. Son uno de los tipos más importantes.
```

...

```
|=====| 90%  
| Un dataframe representa una tabla de datos. Cada columna de éste puede ser de un tipo diferente, pero cada fila debe tener la misma longitud.
```

...

```
|=====| 91%  
| Ahora crea uno. Introduce data.frame(llave=y, color=colores) en la línea de comandos.
```

```
> data.frame(llave=y, color=colores)  
  llave color  
1   561  rojo  
2  1105  azul  
3  1729 verde  
4  2465  azul  
5  2821  rojo
```

| Esa es la respuesta que estaba buscando.

```
|=====| 92%  
| ¿Recuerdas los vectores 'y' y 'colores'? Pues con ellos creaste un dataframe cuya primera columna tiene números de Carmichael y la segunda colores.
```



...

```
|=====| 93%
| Otra manera de crear dataframes es con las funciones read.table() y read.csv().
```

...

```
|=====| 94%
| También puedes usar la función data.matrix() para convertir un dataframe en una matriz.
```

...

```
|=====| 96%
| Antes de concluir la lección, te mostraré un par de atajos.
```

...

```
|=====| 97%
| Al inicio de esta lección introdujiste mi_variable <- (180 / 6) - 15 en la línea de comandos. Supón que cometiste un error y que querías introducir mi_variable <- (180 / 60) - 15, es decir, querías escribir 60, pero escribiste 6. Puedes reescribir la expresión o...
```

...

```
|=====| 98%
| En muchos entornos de programación, presionar la tecla 'flecha hacia arriba' te mostrará comandos anteriores. Presiona esta tecla hasta que llegues al comando (mi_variable <- (180 / 6) - 15), entonces cambia el número 6 por 60 y presiona ENTER. Si la tecla 'flecha hacia arriba' no funciona, sólo escribe el comando correcto.
```

```
> mi_variable <- ((180 / 60) - 15)
```

```
| No exactamente. Dele otra oportunidad. O escribe info() para más opciones.
```

```
| Si la tecla 'flecha hacia arriba' no funciona, sólo escribe el comando correcto.
```

```
> (mi_variable <- (180 / 60) - 15)
[1] -12
```

```
| Intenta de nuevo. ¡De cualquier forma hacerlo bien a la primera es aburrido! O escribe info() para más opciones.
```

```
| Si la tecla 'flecha hacia arriba' no funciona, sólo escribe el comando correcto.
```

```
> mi_variable <- (180 / 60) - 15
```

| ¡Eso es correcto!

```
|=====|
=====| 99%
| Por último, puedes teclear las dos primeras letras del nombre de la variable y después presionar la tecla Tab (tabulador). La mayoría de los entornos de programación muestran una lista de las variables que has creado con el prefijo 'mi_'. Esta función se llama autocompletado y es muy útil para cuando tienes muchas variables en tu espacio de trabajo. Pruébalo, ingresa 'mi_' y autocompleta. Si autocompletar no sirve en tu caso, sólo ingresa mi_variable en la línea de comandos).
```

```
> data.frame(llave=y, color=colores)
```

```
llave color
1 561 rojo
2 1105 azul
3 1729 verde
4 2465 azul
5 2821 rojo
```

| Un buen intento, pero no es exactamente lo que yo esperaba. Inténtalo de nuevo. O escribe info() para más opciones.

| Si autocompletar no sirve en tu caso, sólo introduce mi\_variable en la línea de comandos.

```
> mi_variable
```

```
[1] -12
```

| ¡Mantén este buen nivel!

```
|=====|
=====| 100%
| Has concluido la lección. ¿Te gustaría que se le notificará a Coursera que has completado esta lección?
```

1: No

2: Sí

```
selection: 1
```

| ¡Lo estás haciendo muy bien!

| ¡Has alcanzado el fin de esta lección! volviendo al menú principal...

| Por favor escoge un curso o teclea 0 para salir de swirl.

Sección3

selecciona una lección por favor, o teclea 0 para volver al menú del curso.

1: Obtener Ayuda	2: Objetos Tipos de Datos y Operaciones Basicas
3: Subconjuntos de Datos	4: Leer y escribir Datos
5: Funciones	6: Funciones apply
7: Graficacion	8: Parametros en el Sistema de Graficos
9: Colores en el Sistema de Graficos	10: Graficacion con texto y notacion matematica
11: Creacion de Graficas en 3D	12: Expresiones regulares
13: Graficacion con ggplot2	14: Simulacion

selection: 3

|  
| 0%

| En esta lección conocerás las maneras de acceder a las estructuras de datos en el lenguaje R.

...

|==  
| 2%

| R tiene una sintaxis especializada para acceder a las estructuras de datos.

...

|====  
| 4%

| Tú puedes obtener un elemento o múltiples elementos de una estructura de datos usando la notación de indexado de R.

...

|=====  
| 5%

| R provee diferentes maneras de referirse a un elemento (o conjunto de elementos) de un vector. Para probar estas diferentes maneras crea una variable llamada 'mi\_vector' que contenga un vector con los números enteros del 11 al 30.  
| Recuerda que puedes usar el operador secuencia ':':

> 11:30

```
[1] 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
```

| ¡Casi! Vuelve a intentar de nuevo. O escribe info() para más opciones.

> mi\_vector<- 11:30

| ¡Eso es correcto!

```

|=====
| 7%
| Y ahora ve su contenido.

> mi_vector
[1] 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30

| ¡Excelente trabajo!

|=====
| 9%
| La manera más común de buscar un elemento en R es por medio de un vector
| numérico.

...

|=====
| 11%
| Puedes buscar elementos por posición en un vector usando la siguiente notación: x[s], donde 'x' es un vector del cual
| deseas obtener elementos y 's' es un segundo vector representando el conjunto de índices de elementos que te gustaría
| consultar.

...

|=====
| 12%
| Debes saber que en R las posiciones de los elementos de un vector comienzan en 1 y no en 0, como en lenguajes de
| programación como Java o C.

...

|=====
| 14%
| Puedes usar un vector entero para buscar un simple elemento o múltiples
| .

...

|=====
| 16%
| Por ejemplo, obten el tercer elemento de 'mi_vector'.

> mi_vector[3]
[1] 13

| ¡Buen trabajo!

|=====
| 18%
| Ahora obten los primeros cinco elementos de 'mi_vector'.

> mi_vector[1:5]
[1] 11 12 13 14 15

```

| ¡Eso es trabajo bien hecho!

|=====

| 19%

| No necesariamente los índices deben ser consecutivos. Ingresa mi\_vector[c(4,6,13)] en la línea de comandos.

> mi\_vector[c(4,6,13)]

[1] 14 16 23

| ¡Todo ese trabajo está rindiendo frutos!

|=====

| 21%

| Asimismo, no es necesario que los índices se encuentren ordenados. Ingresa mi\_vector[c(6,13,4)] en la línea de comandos.

> mi\_vector[c(6,13,4)]

[1] 16 23 14

| ¡Mantén este buen nivel!

|=====

| 23%

| Como un caso especial, puedes usar la notación [[]] para referirte a un solo elemento. Ingresa mi\_vector[[3]] en la línea de comandos.

> mi\_vector[[3]]

[1] 13

| ¡Lo has logrado! ¡Buen trabajo!

|=====

| 25%

| La notación [[]] funciona de la misma manera que la notación [] en este caso.

...

|=====

| 26%

| También puedes usar enteros negativos para obtener un vector que consista en todos los elementos, excepto los elementos especificados. Excluye los elementos 9:15, al especificar -9:-15.

> mi\_vector[-9:-15]

[1] 11 12 13 14 15 16 17 18 26 27 28 29 30

| ¡Eres bastante bueno!

|=====

| 28%

| Como alternativa a indexar con un vector de enteros, puedes indexar a través de un vector lógico.

...

```
|=====
| 30%
| Como ejemplo crea un vector lógico de longitud 10 con valores lógicos a
| ternados, TRUE y FALSE (rep(c(TRUE,FALSE),10)),
| y consulta con él mi_vector[rep(c(TRUE,FALSE),10)].
```

```
> mi_vector[rep(c(TRUE,FALSE),10)]
[1] 11 13 15 17 19 21 23 25 27 29
```

| ¡Es asombroso!

```
|=====
| 32%
| Como podrás notar, lo que ocurrió fue que indexaste únicamente los elem
| entos en las posiciones impares, puesto que
| creaste un vector con elementos TRUE en las posiciones impares y FALSE
| en las pares.
```

...

```
|=====
| 33%
| El vector índice no necesita ser de la misma longitud que el vector a i
| ndexar. R repetirá el vector más corto y
| regresará los valores que cacen. Ingresa mi_vector[c(FALSE,FALSE,TRUE)]
| en la línea de comandos.
```

```
> mi_vector[c(FALSE,FALSE,TRUE)]
[1] 13 16 19 22 25 28
```

| ¡Traes una muy buena racha!

```
|=====
| 35%
| Notarás que ahora indexaste los índices de los elementos múltiplos de 3
| .
```

...

```
|=====
| 37%
| Es muy útil calcular un vector lógico de un mismo vector. Por ejemplo,
| busca elementos más grandes que 20. Ingresa en
| la línea de comandos mi_vector > 20.
```

```
> mi_vector > 20
[1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  T
RUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
[20]  TRUE
```

| Perseverancia es la respuesta.

```
|=====
| 39%
```

| Y ahora indexa 'mi\_vector' usando el vector previamente calculado. Ingresamos mi\_vector[(mi\_vector > 20)] en la línea de comandos.

```
> mi_vector[(mi_vector > 20)]  
[1] 21 22 23 24 25 26 27 28 29 30
```

| ¡Traes una muy buena racha!

|=====

| 40%

| También puedes usar esta notación para extraer partes de una estructura de datos multidimensional.

...

|=====

| 42%

| Un arreglo es un vector multidimensional. Vectores y arreglos se almacenan de la misma manera internamente, pero un arreglo se muestra diferente y se accede diferente.

...

|=====

| 44%

| Para crear un arreglo de dimensión 3x3x2 y de contenido los números del 1 al 18 y guardarlo en la variable 'mi\_arreglo', ingresa mi\_arreglo <- array(c(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18),dim=c(3,3,2)) en la línea de comandos.

```
> mi_arreglo <- array(c(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18),dim=c(3,3,2))
```

| ¡Es asombroso!

|=====

| 46%

| Ahora ve el contenido de la variable 'mi\_arreglo'.

```
> mi_arreglo
```

, , 1

	[,1]	[,2]	[,3]
[1,]	1	4	7
[2,]	2	5	8
[3,]	3	6	9

, , 2

	[,1]	[,2]	[,3]
[1,]	10	13	16
[2,]	11	14	17
[3,]	12	15	18

| Esa es la respuesta que estaba buscando.

|=====

| 47%

| R tiene una manera muy limpia de referirse a parte de un arreglo. Se especifican índices para cada dimensión, separados por comas. Ingresas mi\_arreglo[1,3,2] en la línea de comandos.

```
> mi_arreglo[1,3,2]
[1] 16
```

| Perseverancia es la respuesta.

|=====

| 49%

| Asimismo, puedes ingresar mi\_arreglo[1:2,1:2,1] en la línea de comandos. ¡Inténtalo!

```
> mi_arreglo[1:2,1:2,1]
      [,1] [,2]
[1,]     1     4
[2,]     2     5
```

| ¡Eres el mejor!

|=====

| 51%

| Una matriz es simplemente un arreglo bidimensional. Ahora crea una matriz con 3 renglones y 3 columnas con los números enteros del 1 al 9 y guárdala en la variable 'mi\_matriz'.

```
> mi_matriz<- matrix(1:9, nrow = 3,ncol = 3)
```

| ¡Lo has logrado! ¡Buen trabajo!

|=====

| 53%

| Al igual que con los arreglos, para obtener todos los renglones o columnas de una dimensión de una matriz, simplemente omite los índices.

...

|=====

| 54%

| Por ejemplo, si quisieras solo el primer renglón de 'mi\_matriz', basta con ingresar mi\_matriz[1,] en la línea de comandos. ¡Inténtalo!

```
> mi_matriz[1,]
[1] 1 4 7
```

| ¡Lo has logrado! ¡Buen trabajo!

|=====

| 56%

| ¡Ahora obtén solo la primera columna!



```
> mi_matriz[,1]
[1] 1 2 3
```

| ¡Buen trabajo!

```
|=====
| 58%
| También puedes referirte a un rango de renglones. Ingresa mi_matriz[2:3
| ,] en la línea de comandos.
```

```
> mi_matriz[2:3,]
      [,1] [,2] [,3]
[1,]     2     5     8
[2,]     3     6     9
```

| ¡Lo has logrado! ¡Buen trabajo!

```
|=====
| 60%
| O referirte a un conjunto no contiguo de renglones. Ingresa mi_matriz[c
| (1,3),] en la línea de comandos.
```

```
> mi_matriz[c(1,3),]
      [,1] [,2] [,3]
[1,]     1     4     7
[2,]     3     6     9
```

| ¡Eres bastante bueno!

```
|=====
| 61%
| En los ejemplos de arriba solo has visto estructuras de datos basadas e
| n un solo tipo. Recuerda que R tiene un tipo de
| datos incorporado para la mezcla de objetos de diferentes tipos, llamad
| os listas.
```

...

```
|=====
|                                     | 63%
| Debes de saber que en R las listas son sutilmente diferentes de las lis
| tas en muchos otros lenguajes. Las listas en R
| contienen una selección heterogénea de objetos. Puedes nombrar cada com
| ponente en una lista.
```

...

```
|=====
|                                     | 65%
| Los elementos en una lista pueden ser referidos por su ubicación o por
| su nombre.
```

...

```
|=====
|                                     | 67%
```

| Ingresa este ejemplo de una lista con cuatro componentes nombrados carro  
o <- list(color="rojo", nllantas=4, marca=  
| "Renault", ncilindros=4).

```
> carro <- list(color="rojo", nllantas=4, marca="Renault", ncilindros=4)
```

| ¡Excelente trabajo!

|=====| 68%  
=====|  
| Tú puedes acceder a los elementos de una lista de múltiples formas. Pue  
des usar la misma notación que usaste con los  
| vectores.

....

|=====| 70%  
=====|  
| Y además puedes indexar un elemento por nombre usando la notación \$. Po  
r ejemplo, ingresa carro\$color en la línea de  
| comandos.

```
> carro$color
```

```
[1] "rojo"
```

| ¡Lo estás haciendo muy bien!

|=====| 72%  
=====|  
| Además, puedes usar la notación [] para indexar un conjunto de elemento  
s por nombre. Ingresa  
| carro[c("ncilindros","nllantas")] en la línea de comandos.

```
> carro[c("ncilindros","nllantas")]
```

```
$ncilindros
```

```
[1] 4
```

```
$nllantas
```

```
[1] 4
```

| ¡Eso es correcto!

|=====| 74%  
=====|  
| También puedes indexar por nombre usando la notación [[]] cuando selecc  
ionas un simple elemento. Por ejemplo, ingresa  
| carro[["marca"]] en la línea de comandos.

```
> carro[["marca"]]
```

```
[1] "Renault"
```

| ¡Eso es trabajo bien hecho!

|=====| 75%  
=====|

| Hasta puedes indexar por nombre parcial usando la opción exact=FALSE. Ingresa carro[["mar",exact=FALSE]] en la línea de comandos.

```
> carro[["mar",exact=FALSE]]
[1] "Renault"
```

| ¡Bien hecho!

```
|=====
|                                     | 77%
| Ahora crea la siguiente lista: camioneta <- list(color="azul", nllantas
=4, marca= "BMW", ncilindros=6).
```

```
> camioneta <- list(color="azul", nllantas=4, marca= "BMW", ncilindros=6)
```

| ¡Toda esa práctica está rindiendo frutos!

```
|=====
|                                     | 79%
| Algunas veces una lista será una lista de listas. Ingresa cochera <- li
st(carro, camioneta).
```

```
> cochera <- list(carro, camioneta)
```

| ¡Todo ese trabajo está rindiendo frutos!

```
|=====
|                                     | 81%
| Ahora ve el contenido de 'cochera'.
```

```
> cochera
[[1]]
[[1]]$color
[1] "rojo"

[[1]]$nllantas
[1] 4

[[1]]$marca
[1] "Renault"

[[1]]$ncilindros
[1] 4
```

```
[[2]]
[[2]]$color
[1] "azul"

[[2]]$nllantas
[1] 4

[[2]]$marca
[1] "BMW"

[[2]]$ncilindros
```

```
[1] 6
```

```
| ¡Muy bien!
```

```
|=====| 82%  
| Tú puedes usar la notación [[]] para referirte a un elemento en este ti  
po de estructura de datos. Para hacer esto usa  
| un vector como argumento. R iterará a través de los elementos en el vec  
tor referenciando sublistas.
```

```
...
```

```
|=====| 84%  
| Ingresar cochera[[c(2, 1)]] en la línea de comandos.
```

```
> cochera[[c(2, 1)]]  
[1] "azul"
```

```
| ¡Bien hecho!
```

```
|=====| 86%  
| Recuerda que los data frames son una lista que contiene múltiples vecto  
res nombrados que tienen la misma longitud. A  
| partir de este momento usarás el data frame cars del paquete datasets.  
No te preocupes, este paquete viene cargado por  
| defecto.
```

```
...
```

```
|=====| 88%  
| Los datos que conforman al data frame cars son un conjunto de observaci  
ones tomadas en la década de 1920; estas  
| observaciones describen la velocidad (mph) de algunos carros y la dista  
ncia (ft) que les tomó parar.
```

```
...
```

```
|=====| 89%  
| ve el contenido del data frame cars. Ingresas cars en la línea de comand  
os.
```

```
> cars  
  speed dist  
1     4    2  
2     4   10  
3     7    4  
4     7   22  
5     8   16  
6     9   10  
7    10   18
```

8	10	26
9	10	34
10	11	17
11	11	28
12	12	14
13	12	20
14	12	24
15	12	28
16	13	26
17	13	34
18	13	34
19	13	46
20	14	26
21	14	36
22	14	60
23	14	80
24	15	20
25	15	26
26	15	54
27	16	32
28	16	40
29	17	32
30	17	40
31	17	50
32	18	42
33	18	56
34	18	76
35	18	84
36	19	36
37	19	46
38	19	68
39	20	32
40	20	48
41	20	52
42	20	56
43	20	64
44	22	66
45	23	54
46	24	70
47	24	92
48	24	93
49	24	120
50	25	85

| ¡Mantén este buen nivel!

```

|=====
=====| 91%
| Te puedes referir a los elementos de un data frame (o a los elementos d
e una lista) por nombre usando el operador $.
| Ingresa cars$speed en la línea de comandos.

```

```

> cars$speed
[1]  4  4  7  7  8  9 10 10 10 11 11 12 12 12 12 13 13 13 13 14 14 14 14
15 15 15 16 16 17 17 17 18 18 18 18 19 19 19 20
[40] 20 20 20 20 22 23 24 24 24 24 25

```

| ¡Todo ese trabajo está rindiendo frutos!

```
|=====
=====| 93%
| Supón que deseas saber a qué velocidad iban los carros a los que les to
mó más de 100 pies (ft) frenar.
```

...

```
|=====
=====| 95%
| Una manera de encontrar valores específicos en un data frame es al usar
un vector de valores booleanos para especificar
| cuál o cuáles elementos regresar de la lista. La manera de calcular el
vector apropiado es así: cars$dist>100.
| ¡Inténtalo!
```

```
> cars$dist>100
[1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FA
LSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[20] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FA
LSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[39] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE TRUE FA
LSE
```

| ¡Lo has logrado! ¡Buen trabajo!

```
|=====
=====| 96%
| Entonces puedes usar ese vector para referirte al elemento correcto. In
gresas cars$speed[cars$dist>100] en la línea de
| comandos.
```

```
> cars$speed[cars$dist>100]
[1] 24
```

| ¡Tu dedicación es inspiradora!

```
|=====
=====| 98%
| Ahora ya sabes cómo acceder a las estructuras de datos.
```

...

```
|=====
=====| 100%
| Has concluido la lección. ¿Te gustaría que se le notificará a Coursera
que has completado esta lección?
```

#### Sección 4

| Selecciona una lección por favor, o teclea 0 para volver al menú del cu  
rso.

1: Obtener Ayuda	2: Objetos Tipos de Datos y Operaciones Basicas
3: Subconjuntos de Datos	4: Leer y escribir Datos
5: Funciones	6: Funciones apply
7: Graficacion	8: Parametros en el Sistema de Graficos
9: Colores en el Sistema de Graficos	10: Graficacion con texto y notacion matematica
11: Creacion de Graficas en 3D	12: Expresiones regulares
13: Graficacion con ggplot2	14: Simulacion

#### Selection: 4

| 0%

| En esta lección conocerás cómo cargar conjuntos de datos en R y guardar estos conjuntos desde R.

...

| ==  
| 2%

| Una de las mejores cosas acerca de R es lo fácil que es añadir información desde otros programas.

...

| =====  
| 4%

| R puede importar conjuntos de datos desde archivos de texto, otros softwares de estadística y hasta hojas de cálculo.

| No es necesario tener una copia local del archivo. Tú puedes especificar la ubicación del archivo desde una url y R buscará el archivo en Internet.

...

| =====  
| 7%

| La mayoría de los archivos que contienen información tienen un formato similar. Generalmente cada línea del archivo representa una observación o registro, por lo que cada línea contiene un conjunto de diferentes variables asociadas con la observación.

...

| =====  
| 9%

| Algunas veces, diferentes variables son separadas por un carácter especial, llamado delimitador. Otra veces las variables son diferenciadas por su ubicación en cada línea.

...

```
|=====
| 11%
| En esta lección trabajarás con el archivo inmiginthalpry.csv el cual co
ntiene la estimación de personas provenientes de
| otros países que llegan a cada uno de los estados de México. Si tienes
suerte, el archivo se mostrará en algún editor;
| de lo contrario búscalo en el subdirectorío swirl_temp de tu directorio
de trabajo y velo en una aplicación separada.
```

(Se ha copiado el archivo inmiginthalpry.csv a la ruta C:/Users/usuario/D  
ocuments/swirl\_temp/inmiginthalpry.csv ).

...

```
|=====
| 13%
| Como podrás notar el primer renglón del archivo contiene los nombres de
las columnas, en este caso los nombres de cada
| una de las variables de la observación; además, el archivo tiene delimi
tada cada variable de la observación por una
| coma.
```

...

```
|=====
| 15%
| Para cargar este archivo a R, debes especificar que el primer renglón c
ontiene los nombres de las columnas y que el
| delimitador es una coma.
```

...

```
|=====
| 17%
| Para hacer esto necesitarás especificar los argumentos header y sep en
la función read.table. Header para especificar
| que el primer renglón contiene los nombres de la columnas (header=TRUE)
y sep para especificar el delimitador
| (sep=",").
```

...

```
|=====
| 20%
| ¡Importa el archivo inmiginthalpry.csv! Ingresar datos <- read.table("sw
irl_temp/inmiginthalpry.csv", header=TRUE,
| sep=",", fileEncoding = "latin1") en la línea de comandos.
```

```
> datos <- read.table("swirl_temp/inmiginthalpry.csv", header=TRUE, sep=",
", fileEncoding = "latin1")
```

```
| ¡Muy bien!
```

```
|=====
| 22%
| Como podrás notar usaste el argumento fileEncoding; esto debido a que d
e no usarlo R no podría importar el archivo,
```



| puesto que la segunda cadena del archivo: año, no es una cadena válida para el tipo de codificación que read.table usa por defecto. Para poder leer el archivo basta con especificar el argumento fileEncoding. De no especificarlo R te indicará que hay un error.

...

|=====

| 24%

| Intenta usar datos\_2 <- read.table("swirl\_temp/inmigintnalpry.csv", header=TRUE, sep=","). Debido a que el archivo inmigintnalpry.csv contiene caracteres especiales como la ñ, R PUEDE MOSTRARTE UN ERROR. Si R te muestra el error, ingresa ok() en la línea de comandos para continuar.

> ok()

| ¡Es asombroso!

|=====

| 26%

| Este error es muy común cuando intentas leer archivos que su contenido está en español; esto se debe a que usa otra codificación para poder abarcar más símbolos que no usan otros idiomas, como en este caso la ñ. Para poder leer archivos que contengan ñ, basta con especificar el argumento fileEncoding, el cual indica la codificación del archivo a importar; en este caso, usarás fileEncoding = "latin1".

...

|=====

| 28%

| Comúnmente las opciones más importantes son sep y header. Casi siempre debes saber el campo separador y si hay un campo header.

...

|=====

| 30%

| Ahora ve lo que contiene 'datos'. Para hacer esto usarás la función view(). Si te encuentras en Rstudio simplemente puedes presionar el nombre de la variable datos en el apartado Entorno ('Environment') y te mostrará su contenido. Presiona la variable datos en Rstudio o ingresa View(datos) en la línea de comandos.

> View(datos)

| Esa es la respuesta que estaba buscando.

|=====

| 33%

| ¡Como podrás notar el archivo contiene 302060 observaciones!

...

|=====

| 35%

| Es importante saber que no solo existe `read.table()`. R además incluye un conjunto de funciones que llaman a `read.table()` con diferentes opciones por defecto para valores como `sep` y `header`, y algunos otros. En la mayoría de los casos encontrarás que puedes usar `read.csv()` para archivos separados por comas o `read.delim()` para archivos delimitados por TAB sin especificar otras opciones.

...

|=====

| 37%

| La mayoría de las veces deberías ser capaz de cargar archivos de texto en R con la función `read.table()`. Pero algunas veces serás proveído con un archivo de texto que no pueda ser leído correctamente con esta función.

...

|=====

| 39%

| Si estás en Europa y usas comas para indicar punto decimal en los números, entonces puedes usar `read.csv2()` y `read.delim2()`.

...

|=====

| 41%

| Una manera de agilizar la lectura de datos es usando el parámetro `colClasses` de la función `read.table()`.

...

|=====

| 43%

| Este parámetro recibe un vector, el cual describe cada uno de los tipos por columna que va a leer. Esto agiliza la lectura debido a que `read.table()` normalmente lee toda la información y después revisa cada una de las columnas, y decide conforme a lo que vio de qué tipo es cada columna, y al indicar el parámetro `colClasses` le dices a la función `read.table()` de qué tipo son los datos que va a ver, con lo que te evitas el chequeo para saber el tipo de cada columna.

...

|=====

| 46%

| Puedes averiguar la clase de las columnas de manera fácil cuando tienes archivos grandes.

...

```
|=====
| 48%
| Lo que puedes hacer es indicarle a read.table() que solo lea los primer
os 100 renglones del archivo; esto lo haces
| indicando el parámetro nrow. Cabe recordar que debes especificar la cod
ificación del archivo, debido a que usa
| caracteres especiales, también que el primer renglón son los nombres de
la columnas y que el delimitador es una coma.
| Ingresas inicial <- read.table("swirl_temp/inmigintnalpry.csv", header=T
RUE, sep=",", fileEncoding = "latin1", nrow =
| 100) en la línea de comandos.
```

```
> inicial <- read.table("swirl_temp/inmigintnalpry.csv", header=TRUE, sep
=",", fileEncoding = "latin1", nrow =100)
```

```
| ¡Mantén este buen nivel!
```

```
|=====
| 50%
| Con esto has conseguido leer las primeras 100 observaciones.
```

...

```
|=====
| 52%
| Después usa la función sapply mandándole como parámetros el objeto inic
ial (el cual contiene las 100 observaciones) y
| la función class(). Ingresas clases <- sapply(inicial, class) en la líne
a de comandos.
```

```
> clases <- sapply(inicial, class)
```

```
| ¡Lo has logrado! ¡Buen trabajo!
```

```
|=====
| 54%
| Con esto lo que conseguiste fue aplicar la función class() a cada una d
e las columnas del objeto inicial. La función
| class() es una función que determina la clase o tipo de un objeto. Ento
nces los tipos de cada una de las columnas
| fueron guardados en el objeto clases.
```

...

```
|=====
| 57%
| Para ver el contenido del objeto 'clases', basta con escribir clases en
la línea de comandos.
```

```
> clases
  renglon      año      ent    id_ent    cvegeo      sexo
edad inmiginthal
"integer"  "integer"  "factor"  "integer"  "integer"  "factor"
"integer"  "numeric"
```

| ¡Eres bastante bueno!

|=====

| 59%

| Por último, con este vector de clases, leerás todo el archivo usando la función read.table, pero pasándole el argumento colClasses. Ingresa datos <- read.table("swirl\_temp/inmigintnalpry.csv", header=TRUE, sep=";", fileEncoding = "latin1", colClasses=clases) en la línea de comandos.

```
> datos <- read.table("swirl_temp/inmigintnalpry.csv", header=TRUE, sep=";", fileEncoding = "latin1",colClasses=clases)
```

| ¡Eso es trabajo bien hecho!

|=====

| 61%

| Como podrás notar el tiempo de lectura mejoró significativamente usando este truco.

...

|=====

= | 63%

| Si deseas guardar objetos, la manera más simple es usando la función save(). Por ejemplo, puedes usar el siguiente comando para salvar el objeto 'datos' y el objeto 'clases' en el archivo o swirl\_temp/datos\_inmigrates.RData. Ingresa save(datos, clases, file="swirl\_temp/datos\_inmigrates.RData") en la línea de comandos.

```
>  
> save(datos, clases, file="swirl_temp/datos_inmigrates.RData")
```

| ¡Lo estás haciendo muy bien!

|=====

=== | 65%

| La función save() escribe una representación externa de los objetos especificados a un archivo señalado. Además, como ya te habrás dado cuenta, tú puedes guardar múltiples objetos en el mismo archivo, tan solo al listarlos en la función save().

...

|=====

===== | 67%

| Es importante notar que en R, las rutas de archivo siempre son especificadas con diagonales ("/"), aun estando en Microsoft windows. Así que para salvar este archivo al directorio "C:\Documents and Settings\Mi Usuario\Mis Documentos\datos\_inmigrates.RData, solo usarías el siguiente comando: save(datos,file="C:/Documents and Settings/Mi Usuario/Mis Documentos/datos\_inmigrates.RData").

...

```
|=====
=====| 70%
| También es importante notar que el argumento file debe ser explícitamente nombrado.
```

...

```
|=====
=====| 72%
| Ahora que has guardado los objetos 'datos' y 'clases' en un archivo, puedes borrarlos. Introduce rm(datos,clases) en la línea de comandos.
```

```
> rm(datos,clases)
```

```
| ¡Eres bastante bueno!
```

```
|=====
=====| 74%
| Y si ahora usas la función ls(), la cual como recordarás muestra qué conjuntos de datos y funciones un usuario ha definido, verás que no están presentes los objetos datos y clases. Ingresa ls() en la línea de comandos.
```

```
> ls()
[1] "c"                "camioneta"          "carro"
"cochera"          "colores"
[6] "complejo"         "display_swirl_file" "fecha_primer_curso_R"
"find_course"      "inicial"
[11] "m"               "mi_arreglo"         "mi_matriz"
"mi_variable"      "mi_vector"
[16] "ncol"            "ok"                 "test1"
"tf"               "y"
```

```
| ¡Traes una muy buena racha!
```

```
|=====
=====| 76%
| Ahora, puedes fácilmente cargar los objetos 'datos' y 'clases' devuelta a R con la función load(). Solo debes especificar el nombre del archivo donde los guardaste. Ingresa load("swirl_temp/datos_inmigrantes.RData") en la línea de comandos.
```

```
> load("swirl_temp/datos_inmigrantes.RData")
```

```
| ¡Tu dedicación es inspiradora!
```

```
|=====
=====| 78%
| Y si ahora usas la función ls(), verás que están presentes los objetos 'datos' y 'clases'. Ingresa ls() en la línea de comandos.
```

```
> ls()
```

[1] "c"	"camioneta"	"carro"
"clases"	"cochera"	
[6] "colores"	"complejo"	"datos"
"display_swirl_file"	"fecha_primer_curso_R"	
[11] "find_course"	"inicial"	"m"
"mi_arreglo"	"mi_matriz"	
[16] "mi_variable"	"mi_vector"	"nco1"
"ok"	"test1"	
[21] "tf"	"y"	

| ¡Eso es correcto!

```

|=====
=====| 80%
| Es importante saber que los archivos guardados en R funcionarán en toda
s las plataformas; es decir, archivos guardados
| en Linux funcionarán si son cargados desde windows o Mac OS X.

```

...

```

|=====
=====| 83%
| Si deseas guardar cada uno de los objetos de tu espacio de trabajo (wor
kspace), puedes hacerlo usando la función
| save.image(). De hecho, cuando salgas de la session de R, se te pregunt
ará si deseas salvar tu actual espacio de
| trabajo (workspace). Si señalas que sí lo deseas, tu espacio de trabajo
será guardado de la misma manera que usar esta
| función.

```

...

```

|=====
=====| 85%
| Por último, al igual que para importar datos existe la función read.tab
le(), para exportar datos a un archivo de texto
| existe la función write.table().

```

...

```

|=====
=====| 87%
| Normalmente los datos a exportar son data frames y matrices.

```

...

```

|=====
=====| 89%
| Para exportar un objeto a un archivo basta con escribir la función writ
e.table() y como argumento el nombre del objeto,
| además del nombre del archivo donde se guardará. Ingresa write.table(da
tos, file="swirl_temp/datos.txt") en la línea de
| comandos.

```

```
> write.table(datos, file="swirl_temp/datos.txt")
```

| ¡Mantén este buen nivel!

```
|=====
=====| 91%
| Si tienes suerte, te mostraré el archivo datos.txt en algún editor; de
lo contrario, búscalo en el subdirectorio
| swirl_temp de tu directorio de trabajo y velo en una aplicación separad
a.
```

...

```
|=====
=====| 93%
| Como podrás notar el archivo datos.txt no es igual al archivo inmigintrn
alpry.csv que al inicio de esta lección te
| mostré. Una de las principales razones es que para escribir el objeto d
atos no especificaste un delimitador (sep) y por
| defecto R delimitó con espacios.
```

...

```
|=====
=====| 96%
| Al igual que con la función read.table(), R incluye un conjunto de func
iones que llaman a write.table() con diferentes
| opciones por defecto, como lo son write.csv() y write.csv2().
```

...

```
|=====
=====| 98%
| Si deseas, puedes jugar con las funciones write.*() para lograr que dat
os.txt sea identico a inmigintrnalpry.csv.
| Recuerda que para ver los parámetros de write.*() puedes usar help(); p
or ejemplo, help(write.csv).
```

...

```
|=====
=====| 100%
```

## Sección 5

| selecciona una lección por favor, o teclea 0 para volver al menú del cu  
rso.

- |                                      |                         |
|--------------------------------------|-------------------------|
| 1: Obtener Ayuda                     | 2: Objetos Tipos de Da  |
| tos y Operaciones Basicas            |                         |
| 3: Subconjuntos de Datos             | 4: Leer y escribir Dat  |
| os                                   |                         |
| 5: Funciones                         | 6: Funciones apply      |
| 7: Graficacion                       | 8: Parametros en el si  |
| stema de Graficos                    |                         |
| 9: Colores en el sistema de Graficos | 10: Graficacion con tex |
| to y notacion matematica             |                         |
| 11: Creacion de Graficas en 3D       | 12: Expresiones regular |
| es                                   |                         |

Selection: 5

|  
| 0%

| En esta lección conocerás las funciones del lenguaje R.

...

|====  
| 3%

| En R las operaciones que hacen todo el trabajo son llamadas funciones.

...

|=====  
| 6%

| Una función es un objeto en R, que puede tomar como entrada algunos objetos (llamados argumentos de función) y puede regresar un objeto de salida.

...

|=====  
| 9%

| Las mayoría de las funciones son de la siguiente forma: f(argumento\_1, argumento\_2, ...). Donde f es el nombre de la función y argumento\_1, argumento\_2, ... son argumentos para la función.

...

|=====  
| 12%

| Has usado alguna función anteriormente, ya que no se puede hacer nada interesante sin ellas. Todo el trabajo en R es hecho por funciones.

...

|=====  
| 16%

| Una función que has estado usando a lo largo del curso es la función c(), la cual crea un vector de los elementos que le sean pasados como argumentos. Por ejemplo introduce c(1, 03, 2016) en la línea de comandos.

```
>
> c(1, 03, 2016)
[1] 1 3 2016
```

| ¡Tu dedicación es inspiradora!

|=====  
| 19%



| La mayoría de las funciones en R regresan un valor; este valor puede ser calculado con base en el ambiente de la computadora o con base en la entrada (argumentos), como en este caso, en donde el valor regresado es el vector que contiene a 1, 3 y 2016.

...

|=====

| 22%

| Cada inicialización de variables en R, operaciones aritméticas, hasta repetir código en un loop, puede ser escrita como una función.

...

|=====

| 25%

| Las funciones son creadas usando la función especial `function()` y una vez creadas son guardadas como objetos de R de clase tipo `function`.

...

|=====

| 28%

| En la siguiente pregunta se te pedirá que modifiques un script. Las instrucciones de lo que debes hacer se encontrarán en el script. Una vez que hayas acabado de modificar el script, guarda tus cambios e ingresa `submit()` en la línea de comandos y así el script será evaluado. Si después de hacer esto la línea de comandos te dice que lo vuelvas a intentar y el script nuevamente aparece, esto se debe a que debes corregir tu script, siéntete libre de hacerlo, solo no olvides ingresar `submit()` cada vez que guardes tus cambios.

...

|=====

| 31%

| Generalmente el cuerpo de la función es encerrado entre llaves `{}`, pero no es necesario si el cuerpo es una simple expresión. Por ejemplo, la expresión `sucesor <- function(x) x+1` es equivalente a la que se encuentra en el script.

```
> sucesor <- function(x) x+1
>
> sucesor <- function(x) x+1
> sucesor <- {function(x) x+1}
> {sucesor <- function(x) x+1}
> return(x(x + 1))
Error in x(x + 1) : could not find function "x"
> sucesor <- function(x) {
+   return(x(x + 1))
+ }
> submit()
```

| Leyendo tu script...

| Un buen intento, pero no es exactamente lo que yo esperaba. Inténtalo de nuevo.

| Asegúrate de haber borrado el símbolo # enfrente de la x, para que la última expresión sea  $x + 1$ .

```
> sucesor <- function(x) {  
+   return((x + 1))  
+ }  
> submit()
```

| Leyendo tu script...

| Un buen intento, pero no es exactamente lo que yo esperaba. Inténtalo de nuevo.

| Asegúrate de haber borrado el símbolo # enfrente de la x, para que la última expresión sea  $x + 1$ .

```
> sucesor <- function(x) {  
+   (x + 1)  
+ }  
> submit()
```

| Leyendo tu script...

| Una vez más. ¡Tú puedes hacerlo!

| Asegúrate de haber borrado el símbolo # enfrente de la x, para que la última expresión sea  $x + 1$ .

```
> sucesor <- function(x) {  
+   (x + 1)  
+ }  
> sucesor(5)  
[1] 6  
> submit()
```

| Leyendo tu script...

| Por poco era correcto, sigue intentándolo.

| Asegúrate de haber borrado el símbolo # enfrente de la x, para que la última expresión sea  $x + 1$ .

```
> submit()
```

| Leyendo tu script...

| ¡Traes una muy buena racha!

```
|=====
| 34%
| ¡Ahora que has creado tu primera función ¡pruébala! Ingresa sucesor(5)
| en la línea de comandos. Si tu función funciona,
| debería de regresar únicamente el valor 6.
```

```
> sucesor(5)
[1] 6
```

```
| ¡Traes una muy buena racha!
```

```
|=====
| 38%
| ¡Felicidades!, has escrito tu primera función.
```

```
...
```

```
|=====
| 41%
| Es importante que sepas que si deseas ver el código fuente de cualquier
| función, solo debes de teclear el nombre de la
| función sin argumentos ni paréntesis. Ahora ve el código fuente de la f
| unción que acabas de crear. Ingresa sucesor en
| la línea de comandos.
```

```
>
> sucesor()
Error in sucesor() : argument "x" is missing, with no default
> sucesor
function(x) {
  (x + 1)
}
<bytecode: 0x00000000241977b8>
```

```
| ¡Eso es trabajo bien hecho!
```

```
|=====
| 44%
| La definición de una función en R incluye los nombres de los argumentos
| , como en el caso anterior que nombraste a
| 'x'. Si especificas un valor por defecto para un argumento, entonces el
| argumento será considerado opcional.
```

```
...
```

```
|=====
| 47%
| Ahora harás un función ligeramente más complicada, donde usarás argumen
| tos por defecto. Crearás una función llamada
| diferencia_cuadrada(). Recuerda que para elevar un número a cierta pote
| ncia se usa el operador binario '^'. Asegúrate
| de guardar tus cambios antes de ingresar submit() en la línea de comand
| os.
```

```
> diferencia_cuadrada <- function(x, y) {
+   return (fun(x,y))
+ }
```

```
+  
+ }  
> submit()
```

| Leyendo tu script...

| ¡No tan bien, pero estás aprendiendo! Intenta de nuevo.

| Recuerda establecer el valor por defecto adecuado.

```
> diferencia_cuadrada <- function(x, 2) {  
Error: unexpected numeric constant in "diferencia_cuadrada <- function(x,  
2"  
>   return (fun(x,2))  
Error in fun(x, 2) : could not find function "fun"  
>  
> }  
Error: unexpected '}' in "}"  
> diferencia_cuadrada <- function(x, 2) {  
Error: unexpected numeric constant in "diferencia_cuadrada <- function(x,  
2"  
>   return (fun(x,2))  
Error in fun(x, 2) : could not find function "fun"  
> }  
Error: unexpected '}' in "}"  
> diferencia_cuadrada <- function(x,y=2) {  
+   x**2-y**2  
+ }  
> submit()
```

| Leyendo tu script...

```
Error in source(e$script_temp_path, encoding = "UTF-8") :  
  C:\Users\usuario\AppData\Local\Temp\RtmpmsyKEI/diferencia_cuadrada.R:31  
:36: unexpected numeric constant  
30:  
31: diferencia_cuadrada <- function(x, 2  
                                ^
```

| ¡Eres el mejor!

| =====  
| 50%

| Ahora prueba tu función diferencia\_cuadrada(). Ingresa diferencia\_cuadrada(3) en la línea de comandos.

```
> diferencia_cuadrada(3)  
[1] 5
```

| ¡Sigue trabajando de esa manera y llegarás lejos!

| =====  
| 53%

| ¿Qué ha pasado? Como proveíste un solo argumento a la función, R cazó el argumento a 'x', debido a que 'x' es el

| primer argumento. Por lo que 'y' usó el valor por defecto que definiste (2).

....

|=====

| 56%

| Recordarás que en un llamada a función puedes sobrescribir los valores por defecto. Así que ahora prueba | diferencia\_cuadrada() con dos argumentos. Ingres a diferencia\_cuadrada(10, 5) en la línea de comandos.

```
> diferencia_cuadrada(10, 5)
[1] 75
```

| ¡Traes una muy buena racha!

|=====

| 59%

| En R puedes explícitamente nombrar a los argumentos. Por ejemplo ingres a diferencia\_cuadrada(y = 10, x = 5) en la línea | de comandos.

```
> diferencia_cuadrada(y = 10, x = 5)
[1] -75
```

| ¡Toda esa práctica está rindiendo frutos!

|=====

| 62%

| Como podrás notar es diferente ingresar diferencia\_cuadrada(10, 5) a di ferencia\_cuadrada(y = 10, x = 5).

...

|=====

==== | 66%

| R también caza parcialmente los argumentos; es decir, ingresar diferenc ia\_cuadrada(10, y = 5) resulta en lo mismo que | ingresar diferencia\_cuadrada(x = 10, y = 5) o diferencia\_cuadrada(10, 5 ).

...

|=====

===== | 69%

| Si no especificas un valor por defecto para un argumento, y si no espec ificas el valor de ese argumento cuando llamas a | la función, obtendrás un error si la función intenta usar ese argumento .

...

|=====

===== | 72%

| Si deseas escribir una función que acepte un número variable de argumen tos, en R puedes usar '...'; para hacer esto se

| especifica '...' en los argumentos de la función.

...

```
|=====
=====| 75%
| Ahora escribirás una función usando '...'. Cerciórate de guardar tus ca
mbios en el script antes de que introduzcas
| submit().
```

```
> chartr("aeio", "4310", tolower(paste(...)))
Error in tolower(paste(...)) : '...' used in an incorrect context
> numeros_por_vocales <- function(...){
+   chartr("aeio", "4310", tolower(paste(...)))
+
+ }
> submit()
```

| Leyendo tu script...

| ¡Muy bien!

```
|=====
=====| 78%
| Ahora prueba tu función numeros_por_vocales. Usa la función numeros_por
_vocales pasándole como argumentos las cadenas
| que desees.
```

```
> numeros_por_vocales("asdsad")
[1] "4sds4d"
```

| ¡Toda esa práctica está rindiendo frutos!

```
|=====
=====| 81%
| Muchas funciones en R pueden recibir otras funciones como argumentos. P
or ejemplo, si deseas saber los argumentos de
| una función puedes hacer uso de las funciones args() o formals(), las c
uales reciben como argumento el nombre de la
| función de la que deseas conocer los argumentos.
```

...

```
|=====
=====| 84%
| Ahora muestra los argumentos de la función mean(), la cual regresa el p
romedio de los elementos que recibe como
| argumentos. Usa cualquiera de la funciones antes mencionadas.
```

```
> args(mean)
function (x, ...)
NULL
```

| ¡Toda esa práctica está rindiendo frutos!

```
|=====
=====| 88%
| Es importante que sepas que la función args() es usada principalmente d
e modo interactivo para imprimir los argumentos
| de una función. Para uso en programación considera mejor usar formals()
|
.
```

...

```
|=====
=====| 91%
| El concepto de pasar funciones como argumentos es muy poderoso. Complet
a la función operador_binario() para ver cómo
| funciona. Recuerda guardar tus cambios en el script antes de que introd
uzcas submit().
```

```
> operador_binario <- function(fun, a, b){
+ fun(a,b).
Error: unexpected symbol in:
"operador_binario <- function(fun, a, b){
  fun(a,b)."
> }
Error: unexpected '}' in "}"
> operador_binario <- function(fun, a, b){
+ fun(a,b)
+ }
> submit
function ()
{
  invisible()
}
<environment: namespace:swirl>
> submit()
```

| Leyendo tu script...

| Perseverancia es la respuesta.

```
|=====
=====| 94%
| Ahora prueba tu función operador_binario(). Ingresa operador_binario(`%/
/%`, 7, 3) en la línea de comandos. Recuerda que
| el operador `%//%` no es más que la división entera en R.
```

```
> operador_binario(`%//%`, 7, 3)
[1] 2
```

| ¡Eso es correcto!

```
|=====
=====| 97%
| Por último, recuerda que todas las funciones en R regresan un valor. Al
gunas funciones en R además hacen otras cosas,
| como cambiar el estado de las variables, graficar, cargar o guardar arc
hivos, o hasta acceder a la red.
```

...

```
|=====
=====| 100%
```

## Sección 6

| selecciona una lección por favor, o teclea 0 para volver al menú del curso.

1: Obtener Ayuda	2: Objetos Tipos de Datos y Operaciones Basicas
3: Subconjuntos de Datos	4: Leer y escribir Datos
5: Funciones	6: Funciones apply
7: Graficacion	8: Parametros en el Sistema de Graficos
9: Colores en el Sistema de Graficos	10: Graficacion con texto y notacion matematica
11: Creacion de Graficas en 3D	12: Expresiones regulares
13: Graficacion con ggplot2	14: Simulacion

selection: 6

```
|
| 0%
```

| En esta lección aprenderás a utilizar a la familia de funciones `*apply()`.

...

```
|==
| 2%
```

| Cuando te encuentras procesando información, una operación común es la de aplicar una función a un conjunto de elementos y regresar un nuevo conjunto de elementos.

...



```

|====
| 3%
| La funciones *apply() que se encuentran en el paquete base de R, actúan
de esta manera sobre diferentes estructuras de
| datos, aplican una función con uno o varios argumentos y regresan otra
estructura de datos.

...

|=====
| 5%
| La primera función que conocerás es la función apply(), la cual opera s
obre arreglos.

...

|=====
| 7%
| Para conocer el uso de esta función ingresa help("apply") en la línea d
e comandos.

>
> help("apply")

| ¡Eres bastante bueno!

|=====
| 8%
| X es un arreglo (puede ser una matriz si la dimensión del arreglo es 2)
. MARGIN es una variable que define cómo la
| función es aplicada, cuando MARGIN=1 se aplica sobre los renglones, cua
ndo MARGIN=2 trabaja sobre las columnas. FUN es
| la función que deseas aplicar y puede ser cualquier función de R, inclu
yendo funciones definidas por ti.

...

|=====
| 10%
| Opcionalmente puedes especificar argumentos a FUN como argumentos adici
onales (...).

...

|=====
| 11%
| Para ejemplificar cómo funciona crea un arreglo bidimensional (matriz)
y guárdalo en la variable 'mi_matriz'. Ingresa
| mi_matriz <- matrix(data=1:16,nrow=4, ncol=4) en la línea de comandos.

>
> mi_matriz <- matrix(data=1:16,nrow=4, ncol=4)

| ¡Todo ese trabajo está rindiendo frutos!

```

```
|=====
| 13%
| Ahora ve su contenido.
```

```
> mi_matriz
      [,1] [,2] [,3] [,4]
[1,]    1    5    9   13
[2,]    2    6   10   14
[3,]    3    7   11   15
[4,]    4    8   12   16
```

```
| ¡Sigue trabajando de esa manera y llegarás lejos!
```

```
|=====
| 15%
| Ahora imagina que deseas saber el mínimo valor presente en cada columna
de 'mi_matriz'. Puedes usar la función min(),
| la cual regresa el mínimo de los valores que recibe como entrada; y así
, al ingresar apply(X=mi_matriz, MARGIN=2,
| FUN=min) en la línea de comandos obtendrás los mínimos valores por colu
mna. ¡Inténtalo!
```

```
> apply(X=mi_matriz, MARGIN=2,FUN=min)
[1] 1 5 9 13
```

```
| ¡Muy bien!
```

```
|=====
| 16%
| Es importante notar que es necesario que MARGIN=2 para que la función m
in() sea aplicada sobre las columnas.
```

```
...
```

```
|=====
| 18%
| Ahora haz la misma operación, solo que sobre los renglones.
```

```
> apply(X=mi_matriz, MARGIN=1,FUN=min)
[1] 1 2 3 4
```

```
| ¡Eso es correcto!
```

```
|=====
| 20%
| Te preguntarás qué puedes hacer si deseas aplicar una función a una lis
ta o un vector, ya que apply() solo opera sobre
| arreglos.
```

```
...
```

```
|=====
| 21%
| Pues bien, para aplicar una función a cada elemento de un vector o una
lista y regresar una lista, puedes usar la
| función lapply().
```

...

```
|=====
| 23%
```

| Para ejemplificar esto crea una lista, la cual contenga a las cadenas "Introducción", "a", "la", "Programación", "Estadística", "con", "R", en ese orden, y guarda la lista en la variable 'mi\_lista'.

```
> mi_lista<-c("Introducción", "a", "la", "Programación","Estadística", "con", "R")
```

| Intenta de nuevo. ¡De cualquier forma hacerlo bien a la primera es aburrido! O escribe info() para más opciones.

| Recuerda que para crear una lista con ciertos elementos simplemente debes usar la función list() y pasarle como argumento los elementos que desees que contenga.

```
> mi_lista<-list("Introducción", "a", "la", "Programación","Estadística", "con", "R")
```

| ¡Buen trabajo!

```
|=====
| 25%
```

| Ahora ve el contenido de la variable 'mi\_lista'.

```
> mi_lista
```

```
[[1]]
[1] "Introducción"
```

```
[[2]]
[1] "a"
```

```
[[3]]
[1] "la"
```

```
[[4]]
[1] "Programación"
```

```
[[5]]
[1] "Estadística"
```

```
[[6]]
[1] "con"
```

```
[[7]]
[1] "R"
```

| ¡Sigue trabajando de esa manera y llegarás lejos!

```
|=====
| 26%
```

| Aunque la función lapply() es muy parecida a la función apply(), es importante conocer la manera de usarla; esto lo

| puedes hacer conociendo sus argumentos. Ahora ve los argumentos de la función lapply().

```
> lapply()
Error in match.fun(FUN) : argument "FUN" is missing, with no default
> args(lapply)
function (X, FUN, ...)
NULL
```

| Perseverancia es la respuesta.

```
|=====
| 28%
| Como podrás notar el uso de lapply() es más simple que el de apply, puesto que no hace uso del argumento MARGIN.
```

...

```
|=====
| 30%
| Ahora ve un simple ejemplo de cómo usar lapply. Ingresa mayusculas <- lapply(mi_lista, toupper) en la línea de comandos.
```

```
>
> ayusculas <- lapply(mi_lista, toupper)
```

| Un buen intento, pero no es exactamente lo que yo esperaba. Inténtalo de nuevo. O escribe info() para más opciones.

```
> mayusculas <- lapply(mi_lista, toupper)
```

| ¡Lo estás haciendo muy bien!

```
|=====
| 31%
| Ahora ve el contenido de 'mayusculas'.
```

```
> mayusculas
[[1]]
[1] "INTRODUCCIÓN"
```

```
[[2]]
[1] "A"
```

```
[[3]]
[1] "LA"
```

```
[[4]]
[1] "PROGRAMACIÓN"
```

```
[[5]]
[1] "ESTADÍSTICA"
```

```
[[6]]
[1] "CON"
```

```
[[7]]  
[1] "R"
```

| ¡Mantén este buen nivel!

```
|=====
| 33%
| Como recordarás, toupper() regresa la cadena que reciba como entrada en
| mayúsculas. Así que lapply simplemente regresa
| una lista que contiene el resultado de aplicar la función toupper() a c
| ada elemento de 'mi_lista'.
```

...

```
|=====
| 34%
| Para verificar que lapply efectivamente regresa una lista ingresa class
| (mayusculas).
```

```
> class(mayusculas)
[1] "list"
```

| ¡Tu dedicación es inspiradora!

```
|=====
| 36%
| Si ahora ingresas lapply(c("Introduccion", "a", "la", "Programacion", "
| Estadistica", "con", "R"), toupper) en la línea
| de comandos, ¿qué crees que ocurra?
```

- 1: Que te mande error
- 2: Que te regrese una lista que contenga las cadenas "INTRODUCCION", "A", "LA", "PROGRAMACION", "ESTADISTICA", "CON", "R"
- 3: Que te regrese un vector que contenga las cadenas "INTRODUCCION", "A", "LA", "PROGRAMACION", "ESTADISTICA", "CON", "R"

selection: 2

| ¡Todo ese trabajo está rindiendo frutos!

```
|=====
| 38%
| Te regresará una lista que contenga a las cadenas "INTRODUCCION", "A",
| "LA", "PROGRAMACION", "ESTADISTICA", "CON", "R"
| pues si recuerdas la función lapply() opera sobre listas y vectores y S
| IEMPRE regresa una lista; es fácil que recuerdes
| esto si piensas que lista + apply = lapply.
```

...

```
|=====
| 39%
| A partir de ahora trabajarás con el archivo ASA_estadisticasPasajeros(3
| ).csv, el cual contiene datos estadísticos del
| año 2015 de pasajeros en servicio nacional e internacional de la Red Ae
| ropuertos y Servicios Auxiliares de México. Con
```

| suerte, el archivo se mostrará en algún editor. De lo contrario, búscalo en el subdirectorio swirl\_temp, de tu directorio de trabajo y vélo en una aplicación separada.

(Se ha copiado el archivo ASA\_estadisticasPasajeros(3).csv a la ruta C:/Users/usuario/Documents/swirl\_temp/ASA\_estadisticasPasajeros(3).csv ).

...

```
|=====
| 41%
| Como podrás notar el primer renglón del archivo contiene los nombres de las columnas. Año mes son representados por una cadena que contiene año y mes pegados sin espacios; Código IATA se refiere a la sigla utilizada por IATA(International Air Transport Association) para identificar al aeropuerto; Descripción contiene la ciudad donde se encuentra dicho aeropuerto; Estado, como su nombre lo indica, contiene el nombre del estado donde se encuentra dicho aeropuerto; Pasajeros nacionales se refiere al número de pasajeros mexicanos que hicieron uso del aeropuerto; y Pasajeros internacionales se refiere al número de pasajeros extranjeros que hicieron uso del aeropuerto.
```

...

```
|=====
| 43%
| Ahora importa el archivo ASA_estadisticasPasajeros(3).csv usando alguna función read.*() y guárdalo en la variable 'asa_datos'. Recuerda que se encuentra en tu directorio de trabajo, en la carpeta swirl_temp, por lo que la ruta del archivo es "swirl_temp/ASA_estadisticasPasajeros(3).csv"
```

```
> read.csv(swirl_temp/ASA_estadisticasPasajeros(3).csv)
Error: unexpected symbol in "read.csv(swirl_temp/ASA_estadisticasPasajeros(3).csv"
> asa_datos<-read.csv(swirl_temp/ASA_estadisticasPasajeros(3).csv)
Error: unexpected symbol in "asa_datos<-read.csv(swirl_temp/ASA_estadisticasPasajeros(3).csv"
> asa_datos<-read.csv("swirl_temp/ASA_estadisticasPasajeros(3).csv")
```

| ¡Muy bien!

```
|=====
| 44%
| Ahora ve lo que contiene 'asa_datos'. Para hacer esto usarás la función view(). Si te encuentras en Rstudio simplemente puedes presionar el nombre de tu variable asa_datos en el apartado Entorno ("Environment") y se mostrará su contenido. Presiona la variable asa_datos en Rstudio o ingresa en la línea de comando: view(asa_datos).
```

```
> view(asa_datos)
```

| ¡Lo has logrado! ¡Buen trabajo!

```
|=====
| 46%
| Como recordarás la familia de funciones read.*() te regresan un data frame.
```

...

```
|=====
| 48%
| Los data frames no son más que una lista de vectores, así que puedes usar la función lapply() con ellos.
```

....

```
|=====
| 49%
| Es importante que cuando trabajes con datos sepas un poco más de ellos.
```

...

```
|=====
| 51%
| Si deseas saber el tipo de variables que contiene cada columna del data frame 'asa_datos' basta con escribir
| lapply(asa_datos, class). Ingresa lapply(asa_datos, class) en la línea de comandos.
```

```
> lapply(asa_datos, class)
```

```
$Anio.mes
[1] "integer"
```

```
$Codigo.IATA
[1] "factor"
```

```
$Descripcion
[1] "factor"
```

```
$Estado
[1] "factor"
```

```
$Pasajeros.nacionales
[1] "integer"
```

```
$Pasajeros.internacionales
[1] "integer"
```

```
| ¡Lo has logrado! ¡Buen trabajo!
```

```
|=====
| 52%
| Como podrás notar, el data frame contiene dos tipos de datos: enteros (Anio.mes, Pasajeros.nacionales y Pasajeros.internacionales) y factores (Codigo.IATA, Descripcion y Estado).
```

...

```
|=====
| 54%
| Como recordarás los factores son una colección ordenada de elementos. S
on usados en R para representar valores
| categóricos. Si haces un poco de memoria recordarás que los valores que
un factor puede tomar se llaman niveles
| ("levels").
```

...

```
|=====
| 56%
| A la hora de trabajar con factores es importante que conozcas los nivel
es que los datos pueden tomar.
```

...

```
|=====
| 57%
| Si deseas conocer los niveles que las columnas del data frame pueden to
mar, los puedes conocer al imprimir dicha
| columna. Por ejemplo, ingresa asa_datos$Descripcion en la línea de coma
ndos para conocer las ciudades que contienen
| aeropuertos pertenecientes a la Red Aeropuertos y Servicios Auxiliares
de México.
```

```
> asa_datos$Descripcion
```

[1] Ciudad Obregon	Colima	Ciudad del Carmen	Campeche
Chetumal	Ciudad Victoria		
[7] Guaymas	Loreto	Matamoros	Nuevo Laredo
Nogales	Poza Rica		
[13] Puebla	Puerto Escondido	Tehuacan	Tepic
Tamuin	Uruapan		
[19] Ciudad Obregon	Colima	Ciudad del Carmen	Campeche
Chetumal	Ciudad Victoria		
[25] Guaymas	Loreto	Matamoros	Nuevo Laredo
Nogales	Poza Rica		
[31] Puebla	Puerto Escondido	Tehuacan	Tepic
Tamuin	Uruapan		
[37] Ciudad Obregon	Colima	Ciudad del Carmen	Campeche
Chetumal	Ciudad Victoria		
[43] Guaymas	Loreto	Matamoros	Nuevo Laredo
Nogales	Poza Rica		
[49] Puebla	Puerto Escondido	Tehuacan	Tepic
Tamuin	Uruapan		
[55] Ciudad Obregon	Colima	Ciudad del Carmen	Campeche
Chetumal	Ciudad Victoria		
[61] Guaymas	Loreto	Matamoros	Nuevo Laredo
Nogales	Poza Rica		
[67] Puebla	Puerto Escondido	Tehuacan	Tepic
Tamuin	Uruapan		
[73] Ciudad Obregon	Colima	Ciudad del Carmen	Campeche
Chetumal	Ciudad Victoria		
[79] Guaymas	Loreto	Matamoros	Nuevo Laredo
Nogales	Poza Rica		



[85]	Puebla	Puerto Escondido	Tehuacan	Tepic
	Tamuin	Uruapan		
[91]	Ciudad Obregon	Colima	Ciudad del Carmen	Campeche
	Chetumal	Ciudad Victoria		
[97]	Guaymas	Loreto	Matamoros	Nuevo Laredo
	Nogales	Poza Rica		
[103]	Puebla	Puerto Escondido	Tehuacan	Tepic
	Tamuin	Uruapan		
[109]	Ciudad Obregon	Colima	Ciudad del Carmen	Campeche
	Chetumal	Ciudad Victoria		
[115]	Guaymas	Loreto	Matamoros	Nuevo Laredo
	Nogales	Poza Rica		
[121]	Puebla	Puerto Escondido	Tehuacan	Tepic
	Tamuin	Uruapan		
[127]	Ciudad Obregon	Colima	Ciudad del Carmen	Campeche
	Chetumal	Ciudad Victoria		
[133]	Guaymas	Loreto	Matamoros	Nuevo Laredo
	Nogales	Poza Rica		
[139]	Puebla	Puerto Escondido	Tehuacan	Tepic
	Tamuin	Uruapan		
[145]	Ciudad Obregon	Colima	Ciudad del Carmen	Campeche
	Chetumal	Ciudad Victoria		
[151]	Guaymas	Loreto	Matamoros	Nuevo Laredo
	Nogales	Poza Rica		
[157]	Puebla	Puerto Escondido	Tehuacan	Tepic
	Tamuin	Uruapan		
[163]	Ciudad Obregon	Colima	Ciudad del Carmen	Campeche
	Chetumal	Ciudad Victoria		
[169]	Guaymas	Loreto	Matamoros	Nuevo Laredo
	Nogales	Poza Rica		
[175]	Puebla	Puerto Escondido	Tehuacan	Tepic
	Tamuin	Uruapan		
[181]	Ciudad Obregon	Colima	Ciudad del Carmen	Campeche
	Chetumal	Ciudad Victoria		
[187]	Guaymas	Loreto	Matamoros	Nuevo Laredo
	Nogales	Poza Rica		
[193]	Puebla	Puerto Escondido	Tehuacan	Tepic
	Tamuin	Uruapan		
[199]	Ciudad Obregon	Colima	Ciudad del Carmen	Campeche
	Chetumal	Ciudad Victoria		
[205]	Guaymas	Loreto	Matamoros	Nuevo Laredo
	Nogales	Poza Rica		
[211]	Puebla	Puerto Escondido	Tehuacan	Tepic
	Tamuin	Uruapan		

18 Levels: Campeche Chetumal Ciudad del Carmen Ciudad Obregon Ciudad Victoria Colima Guaymas Loreto Matamoros ... Uruapan

| ¡Sigue trabajando de esa manera y llegarás lejos!

```
|=====
| 59%
| Como podrás notar esto presentó todos los valores de la columna Descrip
| cion aun estando repetidos. Al final se muestran
| todos los niveles en donde dice "Levels:" pero al ser muchos R mostró a
| lgunos y los demás los omitió usando "...".
```

...

```
|=====
| 61%
| Una manera de poder ver todos los niveles aun cuando sean muchos es usa
ndo la función unique(). La función unique()
| elimina duplicados. Ingresas unique(asa_datos$Descripcion) en la línea d
e comandos.
```

```
> unique(asa_datos$Descripcion)
 [1] Ciudad Obregon      Colima      Ciudad del Carmen Campeche
Chetumal      Ciudad Victoria
 [7] Guaymas      Loreto      Matamoros      Nuevo Laredo
Nogales      Poza Rica
[13] Puebla      Puerto Escondido Tehuacan      Tepic
Tamuin      Uruapan
18 Levels: Campeche Chetumal Ciudad del Carmen Ciudad Obregon Ciudad Vict
oria Colima Guaymas Loreto Matamoros ... Uruapan
```

```
| ¡Muy bien!
```

```
|=====
| 62%
| Ahora ya conoces las ciudades con aeropuertos pertenecientes a la red A
SA.
```

```
...
```

```
|=====
== | 64%
| Repite el proceso pero ahora para conocer los valores de la columna Est
ado.
```

```
> unique(asa_datos$Estado)
 [1] Sonora      Colima      Campeche      Quint
ana Roo      Tamaulipas
 [6] Baja California Sur Veracruz      Puebla      Oaxac
a      Nayarit
[11] San Luis Potosi Michoacan
12 Levels: Baja California Sur Campeche Colima Michoacan Nayarit Oaxaca
Puebla Quintana Roo San Luis Potosi ... Veracruz
```

```
| ¡Eres bastante bueno!
```

```
|=====
=== | 66%
| Ahora deseas saber el número total de pasajeros nacionales que viajaron
en alguno de los aeropuertos pertenecientes a
| la red ASA. Para hacer esto simplemente puedes sumar todos los elemento
s de la columna de Pasajeros.nacionales. Ingresas
| sum(asa_datos$Pasajeros.nacionales) en la línea de comandos.
```

```
> sum(asa_datos$Pasajeros.nacionales)
[1] 2291419
```

```
| Esa es la respuesta que estaba buscando.
```

```
|=====
=====| 67%
| La función sum() regresa la suma de todos los elementos que recibe como
argumentos.
```

...

```
|=====
=====| 69%
| Ahora deseas obtener el número total de pasajeros internacionales.
```

...

```
|=====
=====| 70%
| Podrías repetir la operación que hiciste para obtener el número total d
e pasajeros nacionales...
```

...

```
|=====
=====| 72%
| O podrías usar la función lapply() y automatizar la operación para obte
ner el número total de ambos pasajeros.
```

...

```
|=====
=====| 74%
| Primero crea un subconjunto de asas_datos, ingresa asa_pasajeros <- asa
_datos[,c("Pasajeros.nacionales",
| "Pasajeros.internacionales")] en la línea de comandos. Con esto obtendr
ás las columnas de pasajeros.
```

```
> asa_pasajeros <- asas_datos[,c("Pasajeros.nacionales","Pasajeros.interna
cionales")]
```

| ¡Excelente trabajo!

```
|=====
=====| 75%
| Ahora ve lo que contiene 'asa_pasajeros'. Para hacer esto usarás la fun
ción View(). Si te encuentras en Rstudio
| simplemente puedes presionar el nombre de la variable asa_pasajeros en
el apartado Entorno ("Environment") y se
| mostrará su contenido. Presiona la variable asa_pasajeros en Rstudio o
ingresa View(asa_pasajeros) en la línea de
| comandos.
```

```
> View(asa_pasajeros)
```

| ¡Mantén este buen nivel!

```
|=====
=====| 77%
| Y ahora usa la función lapply() para obtener el número total de pasajer
os de ambas columna. Ingresa
```

| lapply(asa\_pasajeros, sum) en la línea de comandos.

```
> lapply(asa_pasajeros, sum)
```

```
$Pasajeros.nacionales
```

```
[1] 2291419
```

```
$Pasajeros.internacionales
```

```
[1] 176741
```

| ¡Lo estás haciendo muy bien!

```
|=====
=====| 79%
| Esto te dice que se registraron un total de 2 291 419 pasajeros naciona
les y 176 741 pasajeros internacionales.
```

...

```
|=====
=====| 80%
| Como recordarás, las listas son útiles para guardar objetos de múltiple
s clases. Pero en este caso el resultado de
| aplicar sum() fue un entero por columna; podrías guardar el resultado e
n alguna otra estructura.
```

...

```
|=====
=====| 82%
| sapply() te permite simplificar este proceso; sapply funciona como lapp
ly(), pero intenta simplificar la salida a la
| estructura de datos más elemental que sea posible. De ahí su nombre sim
ple + apply = sapply.
```

...

```
|=====
=====| 84%
| Ahora usa sapply() para obtener el número total de pasajeros nacionales
e internacionales de la misma manera que lo
| hiciste con lapply() y guarda el resultado en la variable 'total_pasaje
ros'.
```

```
> total_pasajeros<-sapply(asa_pasajeros, sum)
```

| ¡Todo ese trabajo está rindiendo frutos!

```
|=====
=====| 85%
| Ahora ve el contenido de 'total_pasajeros'.
```

```
> total_pasajeros
```

```
  Pasajeros.nacionales Pasajeros.internacionales
          2291419             176741
```

| ¡Muy bien!

```
|=====
|                                     | 87%
| Como notarás 'total_pasajeros' no es una lista; esto se debe a que a sa
| pply() simplificó el resultado a un vector de
| enteros.
```

...

```
|=====
|                                     | 89%
| En general, si el resultado de lapply() es una lista donde cada element
| o es de longitud 1, sapply() regresará un
| vector. Si el resultado es una lista donde cada elemento es un vector d
| e la misma longitud (> 1), sapply() regresará
| una matriz. Si sapply() no puede arreglárselas, entonces regresará una
| lista, lo cual no sería diferente al resultado
| de usar lapply().
```

...

```
|=====
|                                     | 90%
| Algunas veces encontrarás que los datos proporcionados tienen una granu
| laridad muy fina para el tipo de análisis que
| estás realizando. Por ejemplo supón que deseas saber el número total de
| pasajeros nacionales por estado. Para encontrar
| ese número, tendrás que agrupar los aeropuertos por estado e ir sumando
| los números de pasajeros nacionales. Esta tarea
| puede volverse un poco conflictiva debido a que la información la tiene
| s dividida por meses.
```

...

```
|=====
|                                     | 92%
| Como recordarás, la columna Estados solo puede tomar los valores Sonora
| , Colima, Campeche, Quintana Roo, Tamaulipas,
| Baja California Sur, Veracruz, Puebla, Oaxaca, Nayarit, San Luis Potosí
| y Michoacán. Para ver cuántos registros tiene
| cada estado ingresa table(asa_datos$Estado) en la línea de comandos.
```

```
> table(asa_datos$Estado)
```

Baja California Sur		Campeche		Colima	
Michoacan		Nayarit			
	12		24		12
12		12			
	Oaxaca		Puebla	Quintana Roo	San L
uis Potosi		Sonora			
	12		24		12
12		36			
	Tamaulipas		Veracruz		
	36		12		

```
| ¡Excelente trabajo!
```

```
|=====
===== | 93%
| Como te podrás dar cuenta cada estado tiene un número múltiplo de 12 re
gistros; esto se debe a que los registros por
| aeropuerto están divididos por mes.
```

...

```
|=====
===== | 95%
| Pensando en que algunas veces tendrás la información con una granularid
ad muy fina para el tipo de análisis que deseas
| realizar, R proporciona la función tapply(), la cual divide datos en gr
upos, basados en valor de alguna variable y
| luego aplica la función especificada a los miembros de cada grupo.
```

...

```
|=====
===== | 97%
| Ingresa tapply(asa_datos$Pasajeros.nacionales, asa_datos$Estado, sum) p
ara obtener el número de pasajeros nacionales
| por estado.
```

```
> tapply(asa_datos$Pasajeros.nacionales, asa_datos$Estado, sum)
Baja California Sur      Campeche      Colima
Michoacan      Nayarit      769864      112656
95635      113043
Oaxaca      Puebla      Quintana Roo      San L
uis Potosi      Sonora      267567      179259
1598      253788
Tamaulipas      Veracruz
243126      60575
```

| ¡Acertaste!

```
|=====
===== | 98%
| Ahora obtén el promedio o media de pasajeros nacionales que viajaron po
r mes en cada aeropuerto. Recuerda que para
| calcular la media puedes usar la función mean().
```

```
> tapply(asa_datos$Pasajeros.nacionales, asa_datos$Estado, mean)
Baja California Sur      Campeche      Colima
Michoacan      Nayarit      32077.6667      9388.0000
7969.5833      9420.2500
Oaxaca      Puebla      Quintana Roo      San L
uis Potosi      Sonora      11148.6250      14938.2500
133.1667      7049.6667
Tamaulipas      Veracruz
6753.5000      5047.9167
```

| Intenta de nuevo. ¡De cualquier forma hacerlo bien a la primera es aburrido! O escribe info() para más opciones.

| Ingresa tapply(asa\_datos\$Pasajeros.nacionales, asa\_datos\$Codigo.IATA, mean) en la línea de comandos para conocer la media de pasajeros nacionales que viajaron por mes en cada aeropuerto.

```
> tapply(asa_datos$Pasajeros.nacionales, asa_datos$Codigo.IATA, mean)
      CEN      CLQ      CME      CPE      CTM      CVM
GYM 19867.4167 9388.0000 49262.8333 14892.5000 14938.2500 6124.2500 1063.0
833 1050.1667 8078.9167 6057.3333 218.5000
      PAZ      PBC      PXM      TCN      TPQ      TSL
UPN 5047.9167 22062.9167 15142.1667 234.3333 9420.2500 133.1667 7969.5
833
```

| ¡Eres el mejor!

```
|=====
=====| 100%
```

## Sección 7

| Selecciona una lección por favor, o teclea 0 para volver al menú del curso.

```
1: Obtener Ayuda
2: Objetos Tipos de Datos y Operaciones Basicas
3: Subconjuntos de Datos
4: Leer y escribir Datos
5: Funciones
6: Funciones apply
7: Graficacion
8: Parametros en el Sistema de Graficos
9: Colores en el Sistema de Graficos
10: Graficacion con texto y notacion matematica
11: Creacion de Graficas en 3D
12: Expresiones regulares
13: Graficacion con ggplot2
14: Simulacion
```

Selection: 7

```
|
| 0%
```

| En esta lección conocerás el sistema base de graficación en R.

...

```
|=
| 1%
```

| Si estás familiarizado con Microsoft Excel, encontrarás que R puede generar todas las gráficas con las que estás familiarizado: gráficas de pastel, gráficas de barras, etc.

```

| Además, hay muchos más tipos de gráficas disponibles en R.
...
|===
| 3%
| Para empezar ve las gráficas básicas que se pueden producir. Ingresa de
mo(graphics) en la
| línea de comandos, y después presiona Enter para comenzar y para cambia
r de gráfica. SI
| PRESENTAS ALGÚN ERROR ingresa ok() en la línea de comandos.

>
>
>
> demo(graphics)

      demo(graphics)
      ---- ~~~~~~

Type <Return> to start :

> # Copyright (C) 1997-2009 The R Core Team
>
> require(datasets)
> require(grDevices); require(graphics)
> ## Here is some code which illustrates some of the differences between
> ## R and S graphics capabilities. Note that colors are generally speci
fied
> ## by a character string name (taken from the X11 rgb.txt file) and tha
t line
> ## textures are given similarly. The parameter "bg" sets the backgroun
d
> ## parameter for the plot and there is also an "fg" parameter which set
s
> ## the foreground color.
>
>
>
> x <- stats::rnorm(50)
> opar <- par(bg = "white")
> plot(x, ann = FALSE, type = "n")
Hit <Return> to see next plot: return
> abline(h = 0, col = gray(.90))
> lines(x, col = "green4", lty = "dotted")
> points(x, bg = "limegreen", pch = 21)
> title(main = "Simple Use of Color In a Plot",
+       xlab = "Just a whisper of a Label",
+       col.main = "blue", col.lab = gray(.8),

```



```

+       cex.main = 1.2, cex.lab = 1.0, font.main = 4, font.lab = 3)
> ## A little color wheel.      This code just plots equally spaced hues
in
> ## a pie chart.      If you have a cheap SVGA monitor (like me) you will
1
> ## probably find that numerically equispaced does not mean visually
> ## equispaced. On my display at home, these colors tend to cluster at
> ## the RGB primaries. On the other hand on the SGI Indy at work the
> ## effect is near perfect.
>
> par(bg = "gray")

> pie(rep(1,24), col = rainbow(24), radius = 0.9)
Hit <Return> to see next plot: return

> title(main = "A Sample Color Wheel", cex.main = 1.4, font.main = 3)

> title(xlab = "(Use this as a test of monitor linearity)",
+       cex.lab = 0.8, font.lab = 3)

> ## We have already confessed to having these. This is just showing off
X11
> ## color names (and the example (from the postscript manual) is pretty
"cute".
>
> pie.sales <- c(0.12, 0.3, 0.26, 0.16, 0.04, 0.12)

> names(pie.sales) <- c("Blueberry", "Cherry",
+                      "Apple", "Boston Cream", "Other", "Vanilla Cream")

> pie(pie.sales,
+     col = c("purple","violetred1","green3","cornsilk","cyan","white"))
Hit <Return> to see next plot: return()

> title(main = "January Pie Sales", cex.main = 1.8, font.main = 1)

> title(xlab = "(Don't try this at home kids)", cex.lab = 0.8, font.lab =
3)

> ## Boxplots: I couldn't resist the capability for filling the "box".
> ## The use of color seems like a useful addition, it focuses attention
> ## on the central bulk of the data.
>
> par(bg="cornsilk")

> n <- 10

> g <- gl(n, 100, n*100)

> x <- rnorm(n*100) + sqrt(as.numeric(g))

> boxplot(split(x,g), col="lavender", notch=TRUE)
Hit <Return> to see next plot:

> title(main="Notched Boxplots", xlab="Group", font.main=4, font.lab=1)

```

```

> ## An example showing how to fill between curves.
>
> par(bg="white")

> n <- 100

> x <- c(0,cumsum(rnorm(n)))
> y <- c(0,cumsum(rnorm(n)))

> xx <- c(0:n, n:0)
> yy <- c(x, rev(y))

> plot(xx, yy, type="n", xlab="Time", ylab="Distance")
Hit <Return> to see next plot:

> polygon(xx, yy, col="gray")

> title("Distance Between Brownian Motions")

> ## Colored plot margins, axis labels and titles.    You do need to be
> ## careful with these kinds of effects.           It's easy to go completely
> ## over the top and you can end up with your lunch all over the keyboard.
> ## On the other hand, my market research clients love it.
>
> x <- c(0.00, 0.40, 0.86, 0.85, 0.69, 0.48, 0.54, 1.09, 1.11, 1.73, 2.05
, 2.02)

> par(bg="lightgray")

> plot(x, type="n", axes=FALSE, ann=FALSE)
Hit <Return> to see next plot:

> usr <- par("usr")

> rect(usr[1], usr[3], usr[2], usr[4], col="cornsilk", border="black")

> lines(x, col="blue")

> points(x, pch=21, bg="lightcyan", cex=1.25)

> axis(2, col.axis="blue", las=1)

> axis(1, at=1:12, lab=month.abb, col.axis="blue")

> box()

> title(main= "The Level of Interest in R", font.main=4, col.main="red")

> title(xlab= "1996", col.lab="red")

> ## A filled histogram, showing how to change the font used for the
> ## main title without changing the other annotation.
>
> par(bg="cornsilk")

```

```

> x <- rnorm(1000)

> hist(x, xlim=range(-4, 4, x), col="lavender", main="")
Hit <Return> to see next plot:

> title(main="1000 Normal Random Variates", font.main=3)

> ## A scatterplot matrix
> ## The good old Iris data (yet again)
>
> pairs(iris[1:4], main="Edgar Anderson's Iris Data", font.main=4, pch=19
)
Hit <Return> to see next plot:

> pairs(iris[1:4], main="Edgar Anderson's Iris Data", pch=21,
+       bg = c("red", "green3", "blue")[unclass(iris$Species)])
Hit <Return> to see next plot:

> ## Contour plotting
> ## This produces a topographic map of one of Auckland's many volcanic "
> peaks".
>
> x <- 10*1:nrow(volcano)
> y <- 10*1:ncol(volcano)
> lev <- pretty(range(volcano), 10)
> par(bg = "lightcyan")
> pin <- par("pin")
> xdelta <- diff(range(x))
> ydelta <- diff(range(y))
> xscale <- pin[1]/xdelta
> yscale <- pin[2]/ydelta
> scale <- min(xscale, yscale)
> xadd <- 0.5*(pin[1]/scale - xdelta)
> yadd <- 0.5*(pin[2]/scale - ydelta)
> plot(numeric(0), numeric(0),
+      xlim = range(x)+c(-1,1)*xadd, ylim = range(y)+c(-1,1)*yadd,
+      type = "n", ann = FALSE)
Hit <Return> to see next plot:

> usr <- par("usr")
> rect(usr[1], usr[3], usr[2], usr[4], col="green3")

```

```

> contour(x, y, volcano, levels = lev, col="yellow", lty="solid", add=TRUE)

> box()

> title("A Topographic Map of Maunga Whau", font= 4)

> title(xlab = "Meters North", ylab = "Meters West", font= 3)

> mtext("10 Meter Contour Spacing", side=3, line=0.35, outer=FALSE,
+       at = mean(par("usr")[1:2]), cex=0.7, font=3)

> ## Conditioning plots
>
> par(bg="cornsilk")

> coplot(lat ~ long | depth, data = quakes, pch = 21, bg = "green3")
Hit <Return> to see next plot:

> par(opar)

Type <Return> to start : ok()
Hit <Return> to see next plot: return()
Hit <Return> to see next plot:
Hit <Return> to see next plot:
Hit <Return> to see next plot:
Hit <Return> to see next plot:
Hit <Return> to see next plot:
Hit <Return> to see next plot:
Hit <Return> to see next plot:
Hit <Return> to see next plot:
Hit <Return> to see next plot:
Hit <Return> to see next plot:

| ¡Excelente!

|====
| 4%
| Como pudiste observar, R es muy bueno a la hora de graficar.

...

|=====
| 6%
| El sistema base de gráficos de R cuenta con tres tipos básicos de funciones: funciones de
| alto nivel, funciones de bajo nivel y funciones interactivas.

...}

|=====
| 7%
| Las funciones de alto nivel generan gráficas preestablecidas.

...

```

```
|=====
| 9%
| Las funciones de bajo nivel añaden información a un gráfico existente.
...

|=====
| 10%
| Y las funciones interactivas te permiten de forma interactiva añadir in
formación o extraer
| información de gráficos. Este curso sólo cubrirá las funciones de alto
nivel y bajo nivel.
...

|=====
| 12%
| Comienza explorando las funciones de alto nivel.
...

|=====
| 13%
| Las funciones de alto nivel están diseñadas para generar un gráfico a p
artir de la
| información pasada como argumentos de la función.
...

|=====
| 15%
| La función plot() es una de las funciones de alto nivel más comúnmente
usada.
...

|=====
| 16%
| Comienza a jugar con ella. Ingresa plot(1:5) en la línea de comandos.
> plot(1:5)
Hit <Return> to see next plot:
Hit <Return> to see next plot:

| ¡Excelente trabajo!

|=====
| 18%
| Al ingresar plot(1:5) has graficado cada elemento del vector 1:5 (1, 2,
3, 4, 5) contra la
| posición en dicho vector de cada elemento; es decir, graficaste los pun
tos (1, 1), (2, 2),
| (3, 3), (4, 4) y (5, 5).
...


```

```

|=====
| 19%
| Es importante saber que plot() es una función genérica, por lo que graficará dependiendo
| del objeto que le sea pasado como entrada.
...

|=====
| 21%
| ve qué pasa si ahora introduces plot(c(1, 2, 3), c(4, 5, 6)) en la línea de comandos.
> plot(c(1, 2, 3), c(4, 5, 6))
Hit <Return> to see next plot:
Hit <Return> to see next plot: return()

| ¡Lo has logrado! ¡Buen trabajo!

|=====
| 22%
| Como notarás, esta vez introdujiste dos vectores como entrada, cada uno con tres elementos.
| Entonces la gráfica fue construida tomando un elemento del primer vector (posición x) y un
| elemento del segundo vector (posición y) para construir cada punto, los cuales son: (1, 4),
| (2, 5) y (3, 6).
...

|=====
| 24%
| Conoce más del uso de plot(). Ingresa ?plot en la línea de comandos.
>
> plot()
Error in xy.coords(x, y, xlabel, ylabel, log) :
  argument "x" is missing, with no default
> ?plot

| ¡Eres bastante bueno!

|=====
| 25%
| Como ya te habrás dado cuenta plot() recibe dos argumentos principales.
...

|=====
| 27%
| El primero, x, representa las coordenadas en el eje x de los puntos en la gráfica, o
| alternatively una única estructura para graficar, una función o cualquier objeto de R
| que provea un método para graficar.

```

```

...
|=====
| 28%
| El segundo, y, representa las coordenadas en el eje Y de los puntos de
| la gráfica. Pero
| este argumento es OPCIONAL, pues sólo es necesario si x no es una estru
| ctura apropiada.

...
|=====
| 30%
| Y esto explica por qué puedes graficar si le pasas uno o dos vectores c
| omo entrada. En el
| primer caso (1:5) le envías una estructura apropiada; en el segundo le
| envías las
| coordenadas de los puntos en 'x' y 'y' por medio de dos vectores (c(1,
| 2, 3) y c(4, 5, 6)).

...
|=====
| 31%
| Además, plot() recibe '...' como argumento, pues no en todos los casos
| recibirá los mismos
| argumentos adicionales; esto se debe a que plot() es una función genéri
| ca.

...
|=====
| 33%
| Pero la mayoría de las veces acepta los siguientes argumentos: type, ma
| in, sub, xlab, ylab
| y asp.

...
|=====
| 34%
| type sirve para especificar qué tipo de gráfica debe trazar. Los valore
| s que puede tomar
| son: "p" para puntos, "l" para líneas, "b" para ambas (líneas y punto),
| "c" para la parte
| de líneas que se muestra usando "b", "o" para ambas (líneas y puntos) '
| sobrepuestas', "h"
| para 'histograma' como líneas verticales (o 'alta densidad'), "s" para
| escalonado, "S" para
| otro tipo de escalonamiento y "n" para no graficar.

...
|=====
| 36%
| Como te habrás dado cuenta, cuando graficas plot(c(1, 2, 3), c(4, 5, 6)
| ) el tipo de gráfica

```

| por defecto fue puntos. Ahora ingresa plot(c(1, 2, 3), c(4, 5, 6), type="l") en la línea de comandos para usar líneas.

```
> plot(c(1, 2, 3), c(4, 5, 6), type="l")
```

Hit <Return> to see next plot:

Hit <Return> to see next plot:

| ¡Eso es trabajo bien hecho!

| =====

| 37%

| Ahora grafica el vector 6:10 de forma escalonada.

```
> plot(c(6:10))
```

Hit <Return> to see next plot:

Hit <Return> to see next plot:

| ¡Casi! Vuelve a intentar de nuevo. O escribe info() para más opciones.

```
> plot(c(6:10) type="s")
```

Error: unexpected symbol in "plot(c(6:10) type"

```
> plot(c(6:10), type="s")
```

Hit <Return> to see next plot:

Hit <Return> to see next plot:

| Intenta de nuevo. O escribe info() para más opciones.

```
> plot(c(6:10), type="s")
```

Hit <Return> to see next plot:

Hit <Return> to see next plot:

| No, ¡casi!, pero no. Intenta de nuevo. O escribe info() para más opciones.

```
> plot(c(6:10), type="s")
```

Hit <Return> to see next plot:

Hit <Return> to see next plot:

| Estás muy cerca... ¡Puedo sentirlo! Inténtalo de nuevo. O escribe info() para más opciones.

```
> plot(c(6:10), type="b")
```

Hit <Return> to see next plot:

Hit <Return> to see next plot:

| ¡Casi! Vuelve a intentar de nuevo. O escribe info() para más opciones.

```
> plot(c(6:10), type="s")
```

Hit <Return> to see next plot:

Hit <Return> to see next plot:

| ¡No tan bien, pero estás aprendiendo! Intenta de nuevo. O escribe info() para más opciones.

```
> plot(c(6:10), type="s")
```

Hit <Return> to see next plot:



```

Hit <Return> to see next plot:

| No exactamente. Dele otra oportunidad. O escribe info() para más opciones.

> plot(c(6:10), type="c")
Hit <Return> to see next plot:
Hit <Return> to see next plot:

| ¡No tan bien, pero estás aprendiendo! Intenta de nuevo. O escribe info() para más opciones.

> info()

| Cuando estés en el indicador de R (>):
| -- Teclear skip() te permite saltar la pregunta actual.
| -- Teclear play() swirl te deja experimentar con R; swirl no hará caso de lo que teclees...
| -- HASTA que ingreses nxt(), vas a recuperar la atención de swirl.
| -- Al teclear bye() swirl terminará. Tu progreso se guardará.
| -- Al teclear main() te regresa al menú principal de swirl.
| -- Teclear info() muestra estas opciones de nuevo.

> skip()
Hit <Return> to see next plot:

| Ingresando la siguiente respuesta correcta...

> plot(6:10, type="s")
Hit <Return> to see next plot:

| ¡Tu dedicación es inspiradora!

|=====
| 39%
| Para continuar grafica el vector 1:10 usando ambas (líneas y puntos).

> plot(1:10, type="b")
Hit <Return> to see next plot:
Hit <Return> to see next plot:

| Perseverancia es la respuesta.

|=====
| 40%
| El argumento main establece el título de la gráfica. ¡Pruébalo! Introduce plot(1:5,
| main="Mi gráfica") en la línea de comandos.

> plot(1:5, main="Mi gráfica")
Hit <Return> to see next plot:
Hit <Return> to see next plot:

| ¡Eso es correcto!

|=====
| 42%

```

```

| Análogamente sub establece el subtítulo de la gráfica.
...
|=====
| 43%
| El argumento xlab establece un título para el eje x de la gráfica. Anál
ogamente ylab para
| el eje Y. Establece "x" como título del eje x y "y" como título del eje
Y en la gráfica
| anterior.
>
> plot(1:5, main="Mi gráfica",xlab = "x", ylab = "y")
Hit <Return> to see next plot:
Hit <Return> to see next plot:
Hit <Return> to see next plot:

| ¡Traes una muy buena racha!

|=====
| 45%
| asp se refiere a la proporción x/y.
...
|=====
| 46%
| Algunas veces las funciones estándar para graficar no producirán exacta
mente el tipo de
| gráfica que deseas. En estos casos las funciones para graficar de bajo
nivel pueden ser
| usadas para añadir información extra a la actual gráfica, como lo son p
untos, líneas o
| texto.
...
|=====
| 48%
| Una de ellas es la función points(). Con la función points() tú puedes
graficar puntos
| sobre una gráfica. Ingresa points(c(1, 5), c(4, 2), col="green") en la
línea de comandos
| para graficar los puntos (1, 4) y (5, 2).
> points(c(1, 5), c(4, 2), col="green")

| ¡Eres bastante bueno!

|=====
| 49%
| Como puedes notar, esto puede ser muy útil para añadir un conjunto adic
ional de puntos a
| una gráfica existente. Usualmente con un color diferente o un símbolo d
iferente.

```

```

...
|=====
| 51%
| La mayoría de los argumentos de la función plot() aplican para función
points(), incluyendo
| 'x' y opcionalmente 'y'. Pero los argumentos más útiles son: col para e
specificar el color
| del borde para los puntos a graficar, bg para especificar el color de r
elleno de los puntos
| a graficar, pch para especificar el símbolo que se usará para graficar
al punto.

...
|=====
| 52%
| Otra función muy útil es la función lines(). Ingresas lines(c(1, 4), c(2
, 5), col="yellow")
| en la línea de comandos.

> lines(c(1, 4), c(2, 5), col="yellow")

| ¡Muy bien!

|=====
| 54%
| La función lines() grafica un conjunto de segmentos de línea sobre una
gráfica existente.
| Al igual que la función points(), muchos argumentos de plot() aplican p
ara lines(). Los
| valores de 'x' y 'y' especifican las intersecciones entre los segmentos
de línea.

...
|=====
| 55%
| Para trazar una sola línea a través del área de la gráfica, puedes util
izar la función
| abline(). Por lo general, se llama abline() para dibujar una sola línea
. Por ejemplo,
| ingresa abline(h=3,col="red",lty=2) en la línea de comandos.

>
> abline(h=3,col="red",lty=2)

| ¡Bien hecho!

|=====
| 57%
| Como notarás, graficaste una línea horizontal en y=3. Para graficar una
línea vertical en
| x=3, basta con ingresar abline(v=3,col="red",lty=2) en la línea de coma
ndos.

...

```

```

|=====
| 58%
| También puedes especificar múltiples argumentos y abline() graficará la
s líneas
| especificadas. Por ejemplo, ingresa abline(h=1:5,v=1:5, col="purple") e
n la línea de
| comandos para graficar una cuadrícula de líneas entre 1 y 10.
> abline(h=1:5,v=1:5, col="purple")
| ¡Eso es correcto!
|=====
| 60%
| Aunque si deseas graficar una cuadrícula en tu gráfica, es mejor que us
es la función
| grid().
...
|=====
| 61%
| Hasta ahora sólo has trabajado con datos ficticios. Para hacer esto más
interesante
| trabajarás con datos reales a partir de este momento.
...
|=====
| 63%
| Usarás el famoso conjunto de datos iris, el cual contiene las medidas e
n centímetros de
| longitud y ancho de ambos sépalo y pétalo de tres especies de iris (set
osa, versicolor y
| virginica) con 50 ejemplares cada una.
...
|=====
| 64%
| Para cargar el conjunto de datos iris, ingresa data("iris") en la línea
de comandos.
> data("iris")
| ¡Traes una muy buena racha!
|=====
| 66%
| Como notarás, el objeto iris fue cargado. Averigua qué tipo de objeto e
s iris.
> type(iris)
Error in type(iris) : could not find function "type"
> typeof(iris)
[1] "list"

```

| Un buen intento, pero no es exactamente lo que yo esperaba. Inténtalo de nuevo. O escribe  
| info() para más opciones.

| Ingresa class(iris) en la línea de comandos.

```
> class(iris)
[1] "data.frame"
```

| ¡Todo ese trabajo está rindiendo frutos!

|=====

| 67%

| Como verás, iris es un data frame. Ingresa head(iris) en la línea de comandos para ver las  
| primeras seis líneas de contenido de iris.

```
> head(iris)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4          0.2   setosa
2          4.9         3.0          1.4          0.2   setosa
3          4.7         3.2          1.3          0.2   setosa
4          4.6         3.1          1.5          0.2   setosa
5          5.0         3.6          1.4          0.2   setosa
6          5.4         3.9          1.7          0.4   setosa
```

| ¡Traes una muy buena racha!

|=====

| 69%

| Sepal.Length y Sepal.Width representan la longitud y ancho del sépalos respectivamente.  
| Petal.Length y Petal.Width representan la longitud y ancho del pétalo.  
| Species representa a  
| la especie (setosa, versicolor y virginica).

...

|=====

| 70%

| Como ya sabes, plot() es una función genérica, por lo que también puedes pedirle que te  
| grafique un data frame completo. Por ejemplo, iris; ingresa plot(iris) en la línea de  
| comandos.

```
> plot(iris)
Hit <Return> to see next plot:
Hit <Return> to see next plot:
```

| ¡Muy bien!

|=====

| 72%

| O bien, puedes tomar columnas y graficarla. Ingresa plot(iris\$Petal.Length,

```

| iris$Petal.width) en la línea de comandos.

> plot(iris$Petal.Length,iris$Petal.width)
Hit <Return> to see next plot:
Hit <Return> to see next plot:

| ¡Eres bastante bueno!

|=====
| 73%
| Pero no es necesario graficar pares de puntos.

...

|=====
| 75%
| Un histograma es una representación visual de la distribución de un con
junto de datos.

...

|=====
| 76%
| Ahora usa la función hist() para graficar un histograma de las longitud
es de los pétalos.
| Ingresa hist(iris$Petal.Length, col="red") en la línea de comandos.

> hist(iris$Petal.Length, col="red")
Hit <Return> to see next plot:
Hit <Return> to see next plot:

| ¡Bien hecho!

|=====
| 78%
| La función hist() es una función de alto nivel y recibe un vector de va
lores numéricos y
| grafica un histograma. Un histograma consiste de un eje X y un eje Y, y
varias barras de
| diferentes tamaños. La altura del eje Y te muestra la frecuencia con la
que aparecen los
| valores del eje X en el conjunto de datos.

...

|=====
| 79%
| La forma de un histograma es una de sus características más importantes
, pues te permite
| ver relativamente donde se encuentra sitiada la mayor y menor cantidad
de información. Esto
| te permite encontrar valores atípicos.

...

|=====
| 81%

```

```
| En caso de que no quieras que te grafique frecuencias puedes usar el pa  
rámetro freq = FALSE  
| para que grafique probabilidades. Ingresa hist(iris$Petal.Length, col="red", freq=FALSE) en  
la línea de comandos.
```

```
>  
> hist(iris$Petal.Length, col="red", freq=FALSE)  
Hit <Return> to see next plot:  
Hit <Return> to see next plot:
```

```
| ¡Excelente trabajo!
```

```
|=====
| 82%
| También es posible cambiar el número de celdas del histograma. Para eso  
usa el argumento  
| breaks. Dependiendo del número de celdas especificadas el gráfico puede  
ser de una forma u  
| otra. Ingresa hist(iris$Petal.Length, col="red", breaks=5) en la línea  
de comandos.
```

```
> hist(iris$Petal.Length, col="red", breaks=5)  
Hit <Return> to see next plot:  
Hit <Return> to see next plot:
```

```
| ¡Excelente!
```

```
|=====
| 84%
| Además de histogramas existen las gráficas de caja.
```

```
...
```

```
|=====
|= | 85%
| La función boxplots(), también de alto nivel, genera gráficas de caja.  
Una gráfica de caja  
| es una forma compacta para mostrar la distribución de una variable. La  
caja muestra el  
| rango intercuartil.
```

```
...
```

```
|=====
== | 87%
| Ahora ingresa ?boxplot en la línea de comandos para conocer el uso de l  
a función boxplot().
```

```
> ?boxplot
```

```
| Perseverancia es la respuesta.
```

```
|=====
=== | 88%
| Como ya habrás notado, boxplot() además recibe un argumento 'formula',  
el cual generalmente
```

| es una expresión con una tilde (~), la cual indica la relación entre la  
s variables de  
| entrada. Eso te permite dar como fórmula algo como Sepal.Width ~ Specie  
s para graficar la  
| relación entre el ancho del sépalos y la especie.

...

|=====

===== | 90%

| Ingresa boxplot(Sepal.Width ~ Species, data=iris, col=2:4) en la línea  
de comandos.

>  
> boxplot(Sepal.Width ~ Species, data=iris, col=2:4)  
Hit <Return> to see next plot:  
Hit <Return> to see next plot:

| ¡Tu dedicación es inspiradora!

|=====

===== | 91%

| boxplot() te generó por cada especie (setosa, versicolor y virginica) l  
os valores de  
| dispersión de los anchos del sépalos. La gráfica te muestra que el ancho  
del sépalos de la  
| especie setosa es mucho mayor que el de las demás especies.

...

|=====

===== | 93%

| boxplot() puede ser usado para crear gráficas de caja para variables in  
dividuales o para  
| variables por grupo.

...

|=====

===== | 94%

| Al igual que con hist() puedes usar los mismos argumentos que usaste en  
plot() para añadir  
| títulos (título, subtítulo, eje x, eje y).

...

|=====

===== | 96%

| Además, existen las gráficas de pastel.

...

|=====

===== | 97%

| Las graficas de pastel no son recomendadas ya que sus características s  
on algo limitadas.



| Los autores recomiendan usar gráficas de barras o de puntos en vez de gráficas de pastel,  
 | debido a que las personas son capaces de juzgar longitudes con mayor precisión que  
 | volúmenes.

...

|=====| 99%

| Las gráficas de pastel son creadas con la función `pie(x, labels=)`, donde `x` es un vector  
 | numérico positivo indicando el área de las rebanadas y `labels` es un vector que indica el  
 | nombre de cada rebanada. Ingresar `pie(c(50, 50, 50), labels=levels(iris$Species))` en la  
 | línea de comandos.

```
> pie(c(50, 50, 50), labels=levels(iris$Species))
```

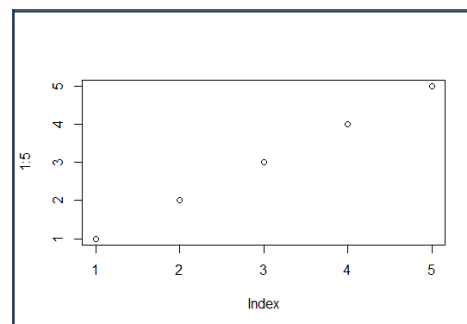
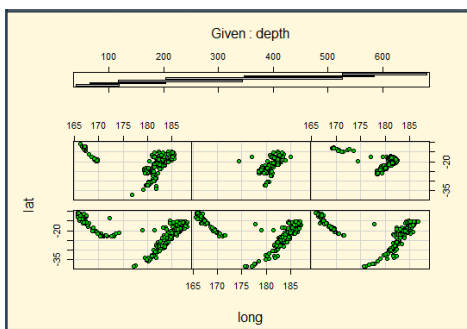
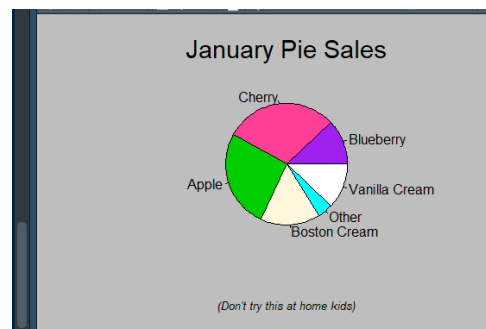
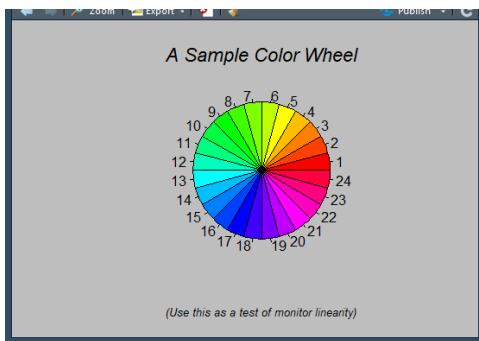
Hit <Return> to see next plot:

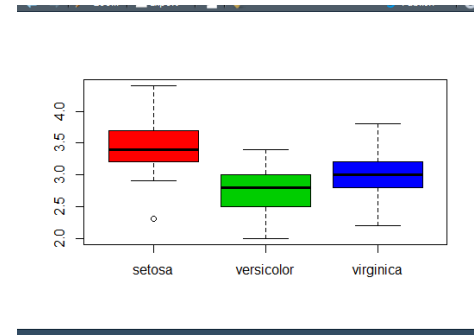
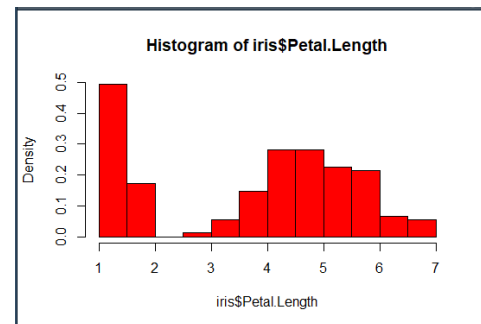
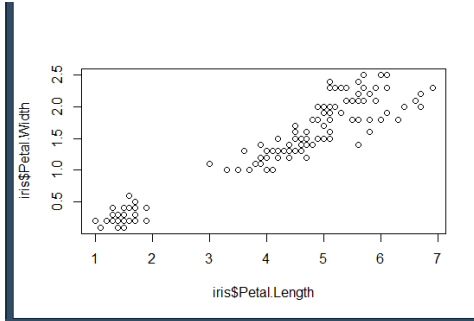
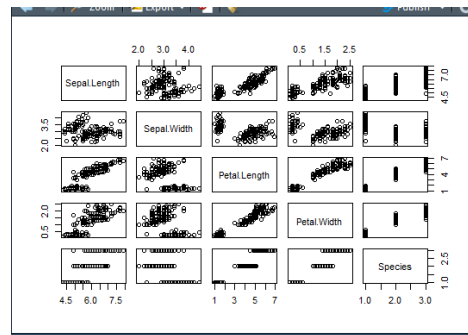
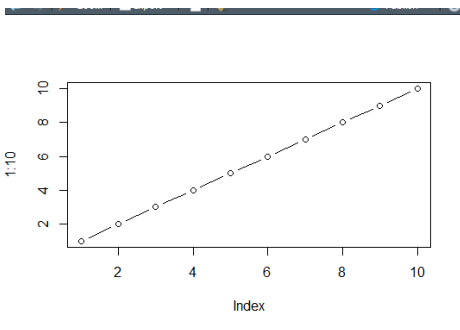
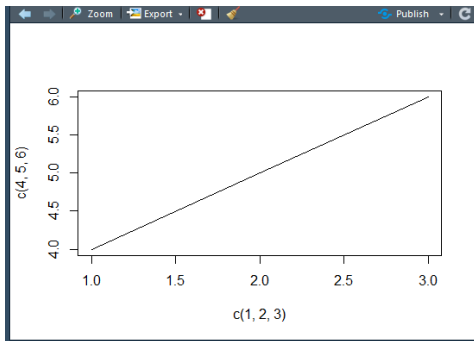
Hit <Return> to see next plot:

| ¡Excelente!

|=====| 100%

Algunos ejemplos de gráficos mostrados en este capítulo de trabajo





## Sección 8

selecciona una lección por favor, o teclea 0 para volver al menú del curso.

- 1: Obtener Ayuda
- 2: Objetos Tipos de Datos y Operaciones Basicas
- 3: Subconjuntos de Datos
- 4: Leer y escribir Datos
- 5: Funciones
- 6: Funciones apply
- 7: Graficacion
- 8: Parametros en el Sistema de Graficos
- 9: Colores en el Sistema de Graficos
- 10: Graficacion con texto y notacion matematica
- 11: Creacion de Graficas en 3D
- 12: Expresiones regulares
- 13: Graficacion con ggplot2
- 14: Simulacion

Selection: 8

|  
| 0%

| En esta lección verás los parámetros del sistema de gráficos a más detalle.

...

|===  
| 3%  
| Cuando creas gráficos, las opciones por defecto de R no siempre producirán exactamente lo que deseas. Sin embargo, puedes modificar casi cualquier aspecto de la gráfica usando los parámetros del sistema de gráficos.

...

|=====  
| 6%  
| Los parámetros del sistema de gráficos pueden establecerse de dos maneras: de forma permanente (que afecta a todas las funciones de gráficos) o temporal (que afecta sólo a la función llamada como lo viste en la lección Graficación).

...

|=====  
| 9%  
| La función par() es usada para acceder y modificar la lista de parámetros de forma permanente.

...

|=====  
| 12%  
| Para ver más detalles introduce ?par en la línea de comandos.

>  
> ?par

```

| ¡Traes una muy buena racha!

|=====
| 16%
| Las nuevas configuraciones establecidas en la función par() serán los
valores por defecto
| para cualquier gráfica nueva hasta que la sesión sea finalizada.

...

|=====
| 19%
| La función par() puede ser muy útil si deseas establecer los parámetro
s una vez y luego
| graficar múltiples veces con ellos.

...

|=====
| 22%
| Puedes checar o establecer los valores de los parámetros de sistema de
gráficos con la
| función par().

...

|=====
| 25%
| Para obtener una lista mostrando todos los parámetros del sistema de g
ráficos, simplemente
| llama a la función par() sin argumentos. ¡Inténtalo!

> par()
$xlog
[1] FALSE

$ylog
[1] FALSE

$adj
[1] 0.5

$ann
[1] TRUE

$ask
[1] TRUE

$bg
[1] "white"

$btty
[1] "o"

$cex
[1] 1

$cex.axis
[1] 1

$cex.lab
[1] 1

$cex.main

```

```
[1] 1.2

$cex.sub
[1] 1

$cin
[1] 0.15 0.20

$col
[1] "black"

$col.axis
[1] "black"

$col.lab
[1] "black"

$col.main
[1] "black"

$col.sub
[1] "black"

$cra
[1] 14.4 19.2

$crt
[1] 0

$csi
[1] 0.2

$cxy
[1] 0.1633957 0.2178609

$din
[1] 5.625000 3.822917

$err
[1] 0

$family
[1] ""

$fg
[1] "black"

$fig
[1] 0 1 0 1

$fin
[1] 5.625000 3.822917

$font
[1] 1

$font.axis
[1] 1

$font.lab
[1] 1

$font.main
[1] 2
```

```
$font.sub
[1] 1

$lab
[1] 5 5 7

$las
[1] 0

$lend
[1] "round"

$lheight
[1] 1

$ljoin
[1] "round"

$lmitre
[1] 10

$lty
[1] "solid"

$lwd
[1] 1

$mai
[1] 1.02 0.82 0.82 0.42

$mar
[1] 5.1 4.1 4.1 2.1

$mex
[1] 1

$mfcol
[1] 1 1

$mfg
[1] 1 1 1 1

$mfrow
[1] 1 1

$mgp
[1] 3 1 0

$mkh
[1] 0.001

$new
[1] FALSE

$oma
[1] 0 0 0 0

$omd
[1] 0 1 0 1

$omi
[1] 0 0 0 0
```

```
$page
[1] TRUE

$pch
[1] 1

$pin
[1] 4.385000 1.982917

$plt
[1] 0.1457778 0.9253333 0.2668120 0.7855041

$ps
[1] 12

$pty
[1] "m"

$smo
[1] 1

$srt
[1] 0

$tck
[1] NA

$tcl
[1] -0.5

$usr
[1] -2.3883 2.3883 -1.0800 1.0800

$xaxp
[1] -2 2 4

$axs
[1] "r"

$axt
[1] "s"

$xpd
[1] FALSE

$yaxp
[1] -1 1 4

$yaxs
[1] "r"

$yxt
[1] "s"

$ylbias
[1] 0.2
```

| ¡Traes una muy buena racha!

|=====

| 28%

| Puedes guardar todos los valores de par() en un objeto. Ingresa par\_or  
ig <- par() en la

```

| Línea de comandos para guardar los valores en la variable par_orig.
>
> par_orig <- par()
| ¡Mantén este buen nivel!
|
|=====
| 31%
| Esto puede ser útil, ya que puedes volver a utilizar estos valores des
pués de haberlos
| modificado.
...
|=====
| 34%
| Si deseas checar el valor de un parámetro con la función par(), usa el
nombre del parámetro
| como argumento (en cadena). Por ejemplo, ingresa par("col") en la líne
a de comandos para
| conocer el valor por defecto del parámetro col (color).
> par("col")
[1] "black"
| ¡Traes una muy buena racha!
|
|=====
| 38%
| Para establecer el valor de un parámetro, basta con usar el nombre del
parámetro como
| argumento de la función par(). Por ejemplo, puedes usar la función par
() para cambiar el
| valor del parámetro col. Ingresa par(col="blue") en la línea de comand
os.
> par(col="blue")
| ¡Traes una muy buena racha!
|
|=====
| 41%
| Para probar que los cambios se han efectuado, en esta lección volverás
a trabajar con el
| conjunto de datos iris. Carga el conjunto de datos iris.
> data("iris")
| ¡Sigue trabajando de esa manera y llegarás lejos!
|
|=====
| 44%
| Ahora prueba que el color por defecto ha sido cambiado a azul. Ingresa
plot(iris) en la
| línea de comandos para verificar esto.
> plot(iris)
Hit <Return> to see next plot:
Hit <Return> to see next plot:
| ¡Es asombroso!

```



```
|=====
| 47%
| Y si revisas nuevamente el valor de col usando par() encontrarás que h
a sido establecido a
| azul ("blue"). Revisa el valor de col.
```

```
> par()
$xlog
[1] FALSE

$ylog
[1] FALSE

$adj
[1] 0.5

$ann
[1] TRUE

$ask
[1] TRUE

$bg
[1] "white"

$btty
[1] "o"

$cex
[1] 1

$cex.axis
[1] 1

$cex.lab
[1] 1

$cex.main
[1] 1.2

$cex.sub
[1] 1

$cin
[1] 0.15 0.20

$col
[1] "blue"

$col.axis
[1] "black"

$col.lab
[1] "black"

$col.main
[1] "black"

$col.sub
[1] "black"

$cra
[1] 14.4 19.2
```

```
$crt
[1] 0

$csi
[1] 0.2

$cxy
[1] 0.03420753 0.10086153

$din
[1] 5.625000 3.822917

$serr
[1] 0

$family
[1] ""

$fg
[1] "black"

$fig
[1] 0 1 0 1

$fin
[1] 5.625000 3.822917

$font
[1] 1

$font.axis
[1] 1

$font.lab
[1] 1

$font.main
[1] 2

$font.sub
[1] 1

$lab
[1] 5 5 7

$las
[1] 0

$lend
[1] "round"

$lheight
[1] 1

$ljoin
[1] "round"

$lmitre
[1] 10

$lty
[1] "solid"

$lwd
```

```
[1] 1

$mai
[1] 1.02 0.82 0.82 0.42

$mar
[1] 5.1 4.1 4.1 2.1

$mex
[1] 1

$mfcol
[1] 1 1

$mfg
[1] 1 1 1 1

$mfrow
[1] 1 1

$mgp
[1] 3 1 0

$mkh
[1] 0.001

$new
[1] FALSE

$oma
[1] 0 0 0 0

$omd
[1] 0 1 0 1

$omi
[1] 0 0 0 0

$page
[1] TRUE

$pch
[1] 1

$pin
[1] 4.385000 1.982917

$plt
[1] 0.1457778 0.9253333 0.2668120 0.7855041

$ps
[1] 12

$pty
[1] "m"

$smo
[1] 1

$srt
[1] 0

$tck
[1] NA
```

```
$tc1
[1] -0.5

$usr
[1] 0 1 0 1

$xaxp
[1] 0 1 5

$axs
[1] "r"

$axt
[1] "s"

$xpd
[1] FALSE

$yaxp
[1] 0 1 5

$ays
[1] "r"

$ayt
[1] "s"

$ylbias
[1] 0.2
```

| No, ¡casi!, pero no. Intenta de nuevo. O escribe info() para más opciones.

| Ingresa par("col") en la línea de comandos.

```
> par("col")
[1] "blue"
```

| ¡Acertaste!

```
|=====
| 50%
```

| Cuando graficas el data frame iris usando plot(iris), los colores mostrados no dicen mucho acerca de las diferentes especies. Así que le puedes decir a R que grafique usando un color diferente por cada una de la especies usando col=iris\$Species. Ingresa plot(iris, col=iris\$Species) en la línea de comandos.

```
> plot(iris,col=iris$Species)
Hit <Return> to see next plot:
Hit <Return> to see next plot:
```

| ¡Bien hecho!

```
|=====
| 53%
```

| Para continuar es importante recordar los nombres de las columnas de iris. ¡Inténtalo!

```
>
```

```

> head(iris)
  Sepal.Length Sepal.width Petal.Length Petal.width Species
1          5.1          3.5          1.4          0.2  setosa
2          4.9          3.0          1.4          0.2  setosa
3          4.7          3.2          1.3          0.2  setosa
4          4.6          3.1          1.5          0.2  setosa
5          5.0          3.6          1.4          0.2  setosa
6          5.4          3.9          1.7          0.4  setosa

| Intenta de nuevo. O escribe info() para más opciones.

> names(iris)
[1] "Sepal.Length" "Sepal.width"  "Petal.Length" "Petal.width"  "Species"

| ¡Lo has logrado! ¡Buen trabajo!

|=====
| 56%
| Comienza viendo las columnas Sepal.Length y Petal.Length. Otra vez es-
| pecifica un color por
| especie. Ingresas plot(iris$Sepal.Length, iris$Petal.Length, col = iris
| $Species) en la línea
| de comandos.

> plot(iris$Sepal.Length, iris$Petal.Length, col = iris$Species)
Hit <Return> to see next plot:
Hit <Return> to see next plot:

| ¡Eso es correcto!

|=====
| 59%
| Los puntos usados para graficar son difíciles de ver. Escoge otros dife-
| rentes.

...

|=====
| 62%
| Un parámetro importante es el símbolo que se usa para graficar puntos;
| éstos se cambian
| usando el parámetro pch. Este parámetro puede recibir valores de dos m-
| aneras.

...

|=====
| 66%
| Una manera es usando un código numérico, donde cada código numérico co-
| rresponde a un
| símbolo. Pruébalo usando pch=15. Ingresas par(pch=15) en la línea de co-
| mandos.

> par(pch=15)

| ¡Es asombroso!

|=====
| 69%
| El código numérico 15 representa cuadrados. Verifica el nuevo valor qu-
| e estableciste.
| Ingresas plot(iris$Sepal.Length, iris$Petal.Length, col = iris$Species)
| en la línea de

```

```
| comandos.
```

```
> plot(iris$Sepal.Length, iris$Petal.Length, col = iris$Species)
Hit <Return> to see next plot:
Hit <Return> to see next plot:
```

| Esa es la respuesta que estaba buscando.

```
|=====
| 72%
| La otra manera es utilizando una cadena. Ingresas plot(iris$Sepal.Length, iris$Petal.Length, col = iris$Species, pch="A") en la línea de comandos.
```

```
>
> lot(iris$Sepal.Length, iris$Petal.Length,col = iris$Species, pch="A")
Error in lot(iris$Sepal.Length, iris$Petal.Length, col = iris$Species, :
could not find function "lot"
> plot(iris$Sepal.Length, iris$Petal.Length,col = iris$Species, pch="A")
Hit <Return> to see next plot:
Hit <Return> to see next plot:
```

| ¡Lo estás haciendo muy bien!

```
|=====
| 75%
| Como te podrás imaginar, cambiaste el valor del parámetro pch temporalmente, al haberlo modificado en la función de plot() y no por medio de la función par(). Si nuevamente graficas sin especificar pch, verás que el símbolo graficado nuevamente es un cuadrado. ¡Verifica esto! Ingresas plot(iris$Sepal.Length, iris$Petal.Length, col = iris$Species) en la línea de comandos.
```

```
> plot(iris$Sepal.Length, iris$Petal.Length, col = iris$Species)
Hit <Return> to see next plot:
Hit <Return> to see next plot:
```

| ¡Eso es trabajo bien hecho!

```
|=====
| 78%
| Los códigos numéricos que puedes usar son los números del 0 al 25. Para conocer los símbolos disponibles ingresa plot(1:26, pch=0:25) en la línea de comandos.
```

```
> plot(1:26, pch=0:25)
Hit <Return> to see next plot:
Hit <Return> to see next plot:
```

| ¡Lo estás haciendo muy bien!

```
|=====
| 81%
| En particular a los símbolos del 21 al 25 les puedes cambiar el color de la orilla y el de relleno. Esto se hace usando los parámetros col y bg. Ingresas plot(iris$Sepal.Length, iris$Petal.Length, col = iris$Species, pch = 21, bg = "blue") en la línea de comandos.
```

```

> plot(iris$Sepal.Length,iris$Petal.Length, col = iris$Species, pch = 21
, bg = "blue")
Hit <Return> to see next plot:
Hit <Return> to see next plot:

| ¡Toda esa práctica está rindiendo frutos!

|=====
== | 84%
| También puedes cambiar el tamaño de los símbolos usando el argumento c
ex. Ingresa
| plot(iris$Sepal.Length, iris$Petal.Length, col = iris$Species, cex = 2
) en la línea de
| comandos.

> plot(iris$Sepal.Length, iris$Petal.Length, col = iris$Species, cex = 2
)
Hit <Return> to see next plot:
Hit <Return> to see next plot:

| ¡Toda esa práctica está rindiendo frutos!

|=====
==== | 88%
| Es difícil distinguir qué color pertenece a qué especie. Lo mejor será
a hacer una leyenda.
| La función legend() puede ser usada para añadir leyendas a las gráfica
s. Ingresa legend(x =
| 4.5, y = 7, legend = levels(iris$Species), col = c(1:3), pch = 16) par
a añadir una leyenda.

> legend(x =4.5, y = 7, legend = levels(iris$Species), col = c(1:3), pch
= 16)

| ¡Lo has logrado! ¡Buen trabajo!

|=====
==== | 91%
| Los parámetros 'x' y 'y' representan las coordenadas que serán usadas
para la leyenda;
| legend representa a las cadenas que aparecerán en la leyenda; col indi
ca el color de las
| líneas o símbolos que aparecerán en la leyenda; y pch indica los carac
teres o símbolos que
| aparecerán en la leyenda.

...

|=====
==== | 94%
| Recuerda que todos los parámetros a los que modificaste su valor tambí
én los puedes cambiar
| directamente desde las funciones para graficar de alto nivel.

...

|=====
==== | 97%
| Por último, si deseas regresar a los valores de los parámetros por omi
sión que tenías en el
| sistema base, basta con llamar a la función par() pasándole como argum
ento el objeto donde

```

| guardaste los valores originales. Regresa a los valores originales usando la función `par()`.

```
> par()
$xlog
[1] FALSE

$ylog
[1] FALSE

$adj
[1] 0.5

$ann
[1] TRUE

$ask
[1] TRUE

$bg
[1] "white"

$btty
[1] "o"

$cex
[1] 1

$cex.axis
[1] 1

$cex.lab
[1] 1

$cex.main
[1] 1.2

$cex.sub
[1] 1

$cin
[1] 0.15 0.20

$col
[1] "blue"

$col.axis
[1] "black"

$col.lab
[1] "black"

$col.main
[1] "black"

$col.sub
[1] "black"

$cra
[1] 14.4 19.2

$crt
[1] 0
```



```
$csi
[1] 0.2

$cxy
[1] 0.1329989 0.6426896

$din
[1] 5.625000 3.822917

$serr
[1] 0

$family
[1] ""

$fg
[1] "black"

$fig
[1] 0 1 0 1

$fin
[1] 5.625000 3.822917

$font
[1] 1

$font.axis
[1] 1

$font.lab
[1] 1

$font.main
[1] 2

$font.sub
[1] 1

$lab
[1] 5 5 7

$las
[1] 0

$lend
[1] "round"

$lheight
[1] 1

$ljoin
[1] "round"

$lmitre
[1] 10

$lty
[1] "solid"

$lwd
[1] 1

$mai
```

```
[1] 1.02 0.82 0.82 0.42

$mar
[1] 5.1 4.1 4.1 2.1

$mex
[1] 1

$mfcol
[1] 1 1

$mfg
[1] 1 1 1 1

$mfrow
[1] 1 1

$mgp
[1] 3 1 0

$mkh
[1] 0.001

$new
[1] FALSE

$oma
[1] 0 0 0 0

$omd
[1] 0 1 0 1

$omi
[1] 0 0 0 0

$page
[1] TRUE

$pch
[1] 15

$pin
[1] 4.385000 1.982917

$plt
[1] 0.1457778 0.9253333 0.2668120 0.7855041

$ps
[1] 12

$pty
[1] "m"

$smo
[1] 1

$srt
[1] 0

$tck
[1] NA

$tc1
[1] -0.5
```

```
$usr  
[1] 4.156 8.044 0.764 7.136
```

```
$xaxp  
[1] 4.5 8.0 7.0
```

```
$xaxs  
[1] "r"
```

```
$xaxt  
[1] "s"
```

```
$xpd  
[1] FALSE
```

```
$yaxp  
[1] 1 7 6
```

```
$yaxs  
[1] "r"
```

```
$yaxt  
[1] "s"
```

```
$ylbias  
[1] 0.2
```

| No exactamente. Dele otra oportunidad. O escribe info() para más opciones.

| Ingresa par(par\_orig) en la línea de comandos para regresar a los valores originales.

```
> par(par_orig)  
warning messages:  
1: In par(par_orig) :  
  el parámetro del gráfico "cin" no puede ser especificado  
2: In par(par_orig) :  
  el parámetro del gráfico "cra" no puede ser especificado  
3: In par(par_orig) :  
  el parámetro del gráfico "csi" no puede ser especificado  
4: In par(par_orig) :  
  el parámetro del gráfico "cxy" no puede ser especificado  
5: In par(par_orig) :  
  el parámetro del gráfico "din" no puede ser especificado  
6: In par(par_orig) :  
  el parámetro del gráfico "page" no puede ser especificado
```

| ¡Acertaste!

```
|=====
=====| 100%  
| Has concluido la lección. ¿Te gustaría que se le notificará a Coursera  
| que has completado  
| esta lección?
```

```
1: No  
2: Si
```

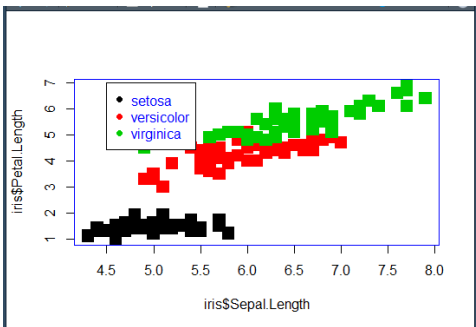
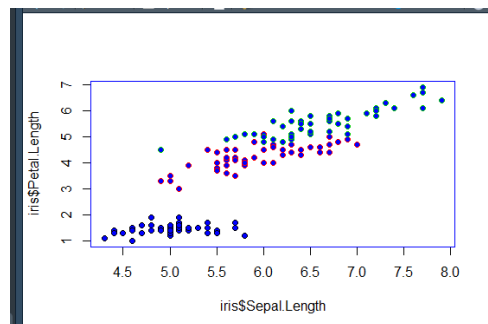
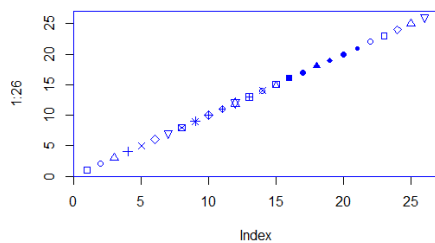
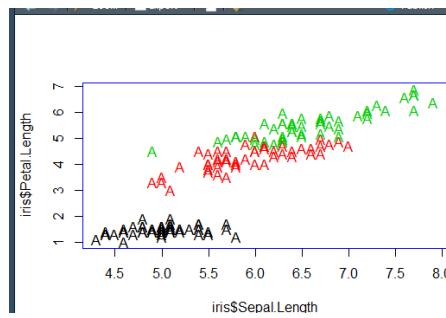
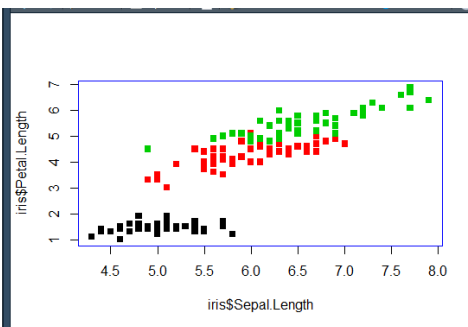
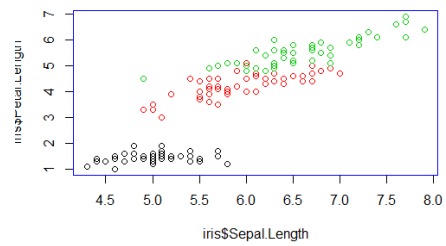
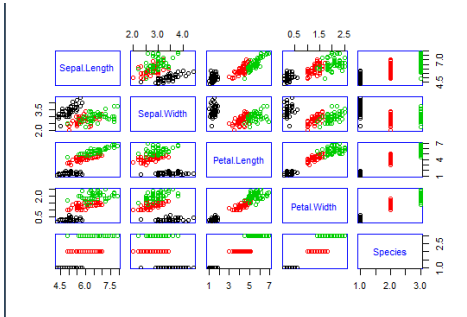
Selection: 1

| ¡Tu dedicación es inspiradora!

| ¡Has alcanzado el fin de esta lección! volviendo al menú principal...  
| Por favor escoge un curso o teclea 0 para salir de swirl.

1: programacion-estadistica-r  
2: ¡Llévame al depósito de cursos swirl!

Algunos graficos logrados con los comandos aprendidos en este capitulo



## Sección 12

| Selecciona una lección por favor, o teclea 0 para volver al  
| menú del curso.

- 1: Obtener Ayuda
- 2: Objetos Tipos de Datos y Operaciones Basicas
- 3: Subconjuntos de Datos
- 4: Leer y escribir Datos
- 5: Funciones
- 6: Funciones apply
- 7: Graficacion
- 8: Parametros en el Sistema de Graficos
- 9: Colores en el Sistema de Graficos
- 10: Graficacion con texto y notacion matematica
- 11: Creacion de Graficas en 3D
- 12: Expresiones regulares
- 13: Graficacion con ggplot2
- 14: Simulacion

Selection: 12

| 0%

| R te provee con algunas funciones que te sirven para  
| manipular cadenas de caracteres.

...

|= 3%

| Recuerda que una cadena contiene todo texto que se  
| encuentre dentro de un par de comillas. Guarda la cadena  
| "¡Hola Mundo!" en la variable 'texto'.

> texto<- "¡Hola Mundo!"

| ¡Traes una muy buena racha!

|=== 5%

| Observa que una cadena es un vector de caracteres, pero no  
| se puede acceder directamente a sus elementos usando la  
| notación de indexado([]). Ingresas texto[1] en la línea de  
| comandos para comprobar esto.

> texto[1]

[1] "¡Hola Mundo!"

| ¡Es asombroso!

|==== 8%

| Y ahora ingresa texto[2] en la línea de comandos.

> texto[2]

[1] NA

| ¡Eso es correcto!

```
|=====| 11%
| Esto se debe a que el acceso a los elementos de una cadena
| se realiza con la función substring(). Para conocer el uso
| de substring() abre su página de ayuda.
```

```
> ?substring()
```

```
| Eso no es precisamente lo que buscaba. Trata otra vez. O
| escribe info() para más opciones.
```

```
| Ingresa ?substring en la línea de comandos para abrir la
| página de ayuda de substring.
```

```
> ?substring
```

```
| ¡Tu dedicación es inspiradora!
```

```
|=====| 13%
| Ahora que conoces su uso, usando la función substring obtén
| la subcadena "Mundo" de la variable texto.
```

```
> substring("Mundo")
Error in substring("Mundo") :
  argument "first" is missing, with no default
> substring("Mundo")
Error in substring("Mundo") :
  argument "first" is missing, with no default
> substring(Mundo)
Error in substring(Mundo) : object 'Mundo' not found
> substring(texto, 7, 11)
[1] "Mundo"
```

```
| ¡Muy bien!
```

```
|=====| 16%
| Pero no sólo eso; esta función puede también utilizarse en asignación,
| lo cual tiene el efecto colateral de cambiar las
| cadenas originales. Para demostrar esto ingresa substring(texto, 7, 11)
| <- "Swirl" en la línea de comandos.
```

```
> substring(texto, 7, 11) <- "Swirl"
```

```
| ¡Muy bien!
```

```
|=====| 18%
| Ahora imprime el contenido de la variable 'texto'.
```

```
> texto
[1] "¡Hola Swirl!"
```

```
| ¡Excelente trabajo!
```

```
|=====| 21%
| Si deseas conocer la longitud de una cadena, no podrás hacer uso de la
| función length(). En cambio deberás usar la
```

| función nchar(). Ingresas nchar(texto) en la línea de comandos para conocer la longitud de texto.

```
> nchar(texto)
```

```
[1] 12
```

| ¡Lo estás haciendo muy bien!

|=====

| 24%

| Anteriormente usaste la función paste(), la cual concatena las cadenas que le sean pasadas como argumento. Ingresas

| paste("¡Adiós", "Mundo!") en la línea de comandos.

```
> paste("¡Adiós", "Mundo!")
```

```
[1] "¡Adiós Mundo!"
```

| ¡Toda esa práctica está rindiendo frutos!

|=====

| 26%

| Debes de saber que por omisión estas cadenas son separadas por un espacio, pero puedes cambiar el comportamiento usando

| el argumento sep. Ingresas paste("¡Adiós", "Mundo!", sep="\_") en la línea de comandos.

```
> paste("¡Adiós", "Mundo!", sep="_")
```

```
[1] "¡Adiós_Mundo!"
```

| ¡Tu dedicación es inspiradora!

|=====

| 29%

| Sin embargo, a menudo es más conveniente para crear una cadena legible usar la función sprintf(), la cual tiene

| sintaxis del lenguaje C.

...

|=====

| 32%

| Por ejemplo, crea la variable 'i' y guarda el número 9 en ella.

```
> i<-9
```

| ¡Excelente trabajo!

|=====

| 34%

| Si quisieras conformar una cadena que te diera la información sobre 'i'. Puedes usar la función sprintf() y usar la

| cadena especial "%d" en cada lugar donde quieras hacer uso de un valor numérico; en este caso 'i'. Ingresas sprintf("El

| cuadrado de %d es %d", i, i^2) en la línea de comandos.

```
> sprintf("Elcuadrado de %d es %d", i, i^2)
```

```
[1] "Elcuadrado de 9 es 81"
```

| Por poco era correcto, sigue intentándolo. O escribe info() para más opciones.

```
> sprintf("El cuadrado de %d es %d", i, i^2)
```

```
[1] "El cuadrado de 9 es 81"
```

```
| ¡Eso es trabajo bien hecho!
```

```
|=====| 37%  
| La función sprintf() ensambla una cadena formateada con los valores que  
vayas pidiendo. Ingresas ?sprintf en la línea de  
| comandos para conocer más acerca de ella.
```

```
> ?sprintf
```

```
| ¡Traes una muy buena racha!
```

```
|=====| 39%  
| Contrario a la función paste() tienes la función strsplit(), la cual su  
bdivide una cadena en cadenas más pequeñas,  
| dependiendo de la cadena indicada como separación. Ingresas strsplit("Me  
/gusta/programar/en/R", "/") en la línea de  
| comandos.
```

```
> strsplit("Me/gusta/programar/en/R", "/")
```

```
[[1]]  
[1] "Me"          "gusta"       "programar"  "en"         "R"
```

```
| ¡Eres el mejor!
```

```
|=====| 42%  
| Anteriormente usaste la función chartr() para sustituir caracteres en c  
adenas. Ingresas chartr("o", "a", "¡Hola Mundo!")  
| en la línea de comandos.
```

```
> chartr("o", "a", "¡Hola Mundo!")
```

```
[1] "¡Ha!a Munda!"
```

```
| ¡Bien hecho!
```

```
|=====| 45%  
| Muchas veces necesitarás reemplazar un texto con otro y chartr() funcio  
nará perfectamente.
```

```
...
```

```
|=====| 47%  
| Pero muchas otras lo que querrás hacer será más complejo, porque puede  
que en vez de ser un simple texto querrás  
| reemplazar todas las palabras que terminen con a.
```

```
...
```

```
|=====| 50%  
| En estos casos puedes utilizar expresiones regulares (que suelen llamar  
se "regex" o "regexp" de forma abreviada).
```

```
...
```

```
|=====| 53%
```



| Las expresiones regulares o patrones no son más que una especie de comodín o un atajo para referirse a una gran cantidad de cadenas.

...

|=====| 55%  
| A diferencia de otros lenguajes, en los que las expresiones regulares se encierran entre algún tipo especial de delimitadores, en R una expresión regular se representa como una cadena de texto.

...

|=====| 58%  
| Por ejemplo, el grupo formado por las cadenas "Handel", "Händel" y "Hae ndel" se describe mediante el patrón  
| "H(a|ä|ae)ndel".

...

|=====| 61%  
| Habitualmente las expresiones regulares se pasan como argumentos de una función, que utiliza el patrón representado por ellas para realizar alguna tarea como búsqueda o sustitución.

...

|=====| 63%  
| Para ejemplificar esto guarda la cadena "H(a|ä|ae)ndel" en la variable 'patron'.

```
> patron<-"H(a|ä|ae)ndel"
```

| ¡Bien hecho!

|=====| 66%  
| Una de las funciones que trabaja con expresiones regulares es grep(). Esta función toma como argumentos primero un patrón y como segundo argumento un vector de cadenas y grep() te regresará un vector numérico, el cual contiene los índices de las cadenas que contienen ese patrón.

...

|=====| 68%  
| Para probar grep() he creado un vector de cadenas y lo guardé en la variable 'musicos'. Revisa el contenido de 'musicos'.

```
> musicos  
[1] "Handel"      "Mendel"      "Haendel"     "HÃxndel"     "Handemore"   "handel"
```

| ¡Bien hecho!

|=====| 71%  
| Ahora ingresa grep(patron, musicos) en la línea de comandos.

```
> grep(patron, musicos)
[1] 1 3
```

| ¡Acertaste!

|=====| 74%  
| Efectivamente grep() te regresó las posiciones de las cadenas que conti  
enen el patrón "H(a|ä|ae)ndel"; es decir, las  
| posiciones de las cadenas "Handel", "Händel" , "Haendel".

...

|=====| 76%  
| Es importante que sepas que el patrón es sensible a mayúsculas; esto ex  
plica por qué no encontré coincidencia en la  
| cadena "handel".

...

|=====| 79%  
| Como ya te habrás imaginado el carácter "|" dentro de una expresión reg  
ular no representa más que un OR; es decir, el  
| patrón te indica a las cadenas: "Handel" o "Händel" o "Haendel".

...

|=====| 82%  
| Si deseas construir un patrón que además incluya las cadenas "Mendel" y  
"handel", puedes hacerlo de la siguiente manera  
| ".(a|ä|ae|e)ndel". Ingresa nuevo\_patron <- ".(a|ä|ae|e)ndel" en la líne  
a de comandos.

```
> nuevo_patron <- ".(a|ä|ae|e)ndel"
```

| ¡Mantén este buen nivel!

|=====| 84%  
| Donde el "." indica un carácter simple, por lo que cuando lo utilizas d  
entro de la expresión regular estás diciendo que  
| no te importa qué carácter está ahí, cualquier carácter cazará con la e  
xpresión regular. Ingresa grep(nuevo\_patron,  
| musicos) para comprobar esto.

```
> grep(nuevo_patron, musicos)
[1] 1 2 3 6
```

| ¡Eso es trabajo bien hecho!

|=====| 87%  
| Otra función que trabaja con expresiones regulares es regexpr(), que al  
igual que grep() recibe como primer argumento  
| un patrón y como segundo un vector de cadenas. A diferencia de grep(),  
regexpr() regresa el índice en donde encuentra  
| la primera aparición del patrón que estás buscando. Prueba la función;  
ingresa regexpr(patron, musicos) en la línea de  
| comandos.

```
> regexpr(patron, musicos)
[1] 1 -1 1 -1 -1 -1
attr(,"match.length")
[1] 6 -1 7 -1 -1 -1
```

| ¡Excelente trabajo!

```
|=====| 89%
| Además de regresar el índice en donde encuentra la primera aparición de
| patrón, también te regresa la longitud del
| patrón encontrado. Si no encuentra el patrón notarás que te regresa com
| o índice y longitud un -1.
```

...

```
|=====| 92%
| Si deseas encontrar todas las posiciones donde es encontrado el patrón
| y no sólo la primera, puedes usar gregexpr(), ya
| que funciona de la misma manera que regexpr(), sólo que te regresa todo
| s los índices donde encuentra el patrón. Por
| ejemplo, ingresa gregexpr(patron, "Georg Friedrich Händel, en inglés Ge
| orge Frideric Handel fue un compositor alemán.")
| en la línea de comandos.
```

```
> gregexpr(patron, "Georg Friedrich Händel, en inglés George Frideric Han
del fue un compositor alemán.")
[[1]]
[1] 17 51
attr(,"match.length")
[1] 6 6
```

| ¡Buen trabajo!

```
|=====| 95%
| Las expresiones regulares pueden ser muy ricas e incluso muy complicada
| s. El objetivo del curso no define que las veas
| a profundidad, pero puedes ver cómo las utiliza R usando ?regexpr. Inté
| ntalo.
```

```
> ??regexpr
```

```
| Por poco era correcto, sigue intentándolo. O escribe info() para más op
| ciones.
```

```
> ?regexpr
```

| ¡Todo ese trabajo está rindiendo frutos!

```
|=====| 97%
| Aquí puedes ver todas las expresiones que te sirven para utilizar expre
| siones regulares dentro del lenguaje con las
| funciones que acabas de ver.
```

...

```
|=====| 100%
```

#### Sección 14

| Selecciona una lección por favor, o teclea 0 para volver al menú del curso.

1: Obtener Ayuda	2: Objetos Tipos de Datos y Operaciones Basicas
3: Subconjuntos de Datos	4: Leer y escribir Datos
5: Funciones	6: Funciones apply
7: Graficacion	8: Parametros en el Sistema de Graficos
9: Colores en el Sistema de Graficos	10: Graficacion con texto y notacion matematica
11: Creacion de Graficas en 3D	12: Expresiones regulares
13: Graficacion con ggplot2	14: Simulacion

Selection: 14

```
|
| 0%
```

| Esta lección asume que tienes familiaridad con algunas distribuciones de probabilidad. Aun si no tienes ninguna experiencia con estos conceptos, serás capaz de completar esta lección y entender las ideas generales.

...

```
|====
| 4%
| R contiene por defecto muchas distribuciones de probabilidad. Por ejemplo, si ingresas ?Normal, obtendrás información acerca de la distribución normal.
```

> ?Normal

| ¡Muy bien!

```
|=====
| 7%
| Usualmente cada distribución tendrá 4 funciones de la forma prefijo + verbo_distribucion. Cada prefijo indica: r - la función para generar números aleatorios, p - la función de distribución, d - la función de densidad, q - la función que te da cuantiles.
```

...

```
|=====
| 11%
| A todas estas funciones les puedes especificar la media y la varianza de la distribución.
```

...

|=====

| 14%

| En general, la primera letra de la función determina qué información obtendrás acerca de la distribución y luego el apodo\_distribución sigue directamente.

...

|=====

| 18%

| Por ejemplo, si quisieras la función de densidad de la distribución Poisson usas dpois(). A continuación se muestran algunas distribuciones útiles.

(Se ha copiado el archivo distributions.txt a la ruta C:/Users/usuario/Documents/swirl\_temp/distributions.txt ).

...

|=====

| 21%

| Para obtener una lista completa de las distribuciones disponibles en R, ingresa help(Distributions) en la línea de comandos.

> help(Distributions)

| ¡Eso es correcto!

|=====

| 25%

| Es importante que revises que las funciones definan la distribución de la manera que tú esperas.

...

|=====

| 29%

| Ahora verás cómo estas funciones te pueden ayudar para resolver un problema de simulación.

...

|=====

| 32%

| Supón que quisieras encontrar la probabilidad de tener al menos 5 caras al lanzar 7 veces una moneda justa.

...

|=====

| 36%

| Recuerda que el lanzamiento de una moneda puede ser modelado por una variable aleatoria bernoulli.

...

```
|=====
| 39%
| Es decir, donde p es la probabilidad de éxito; en este caso que caiga u
na cara, por lo que 1-p es la probabilidad de
| fracaso, es decir, que no caiga cara.
```

...

```
|=====
| 43%
| Y ahora puedes usar una variable aleatoria binomial que te dice el núme
ro de éxitos en n repeticiones de experimentos
| bernoulli. Esta variable aleatoria la puedes modelar con la función rbi
nom(). No olvides que si n=1 el experimento que
| estarás observando es el lanzamiento de una moneda (el caso más simple)
.
```

...

```
|=====
| 46%
| En este caso puedes usar n=7 que son los 7 lanzamientos de la moneda.
```

...

```
|=====
| 50%
| Ahora vas a generar esos números aleatorios usando la función rbinom().
Antes que nada ingresa ?rbinom en la línea de
| comandos para conocer su uso.
```

> ?rbinom

| ¡Acertaste!

```
|=====
| 54%
| Como te podrás dar cuenta rbinom() se encuentra definida así: rbinom(n,
size, prob), donde n indica el número de veces
| que se va a llevar a cabo el experimento. size dice el número de intent
os; en este caso el número de lanzamientos que
| se van a llevar a cabo. Y prob indica la probabilidad de éxito en cada
intento; en este caso que caiga cara.
```

...

```
|=====
| 57%
| Si quisieras llevar a cabo este experimento una única vez, es decir, la
nzar 7 veces una moneda con probabilidad 1/2 de
| éxito (es decir, que la moneda sea justa), deberás llamar a rbinom() de
la siguiente manera: ingresa rbinom(1, 7, 0.5)
| en la línea de comandos.
```

> rbinom(1, 7, 0.5)

```
[1] 5
```

```
| ¡Excelente!
```

```
|=====
| 61%
| El resultado que obtuviste sólo te indica el número de éxitos que obtuv
| iste en un único experimento, es decir, el
| número de caras que salieron.
```

```
...
```

```
|=====
==| 64%
| Si quisieras sacar la proporción de éxitos que tendrías, habría que lle
| var a cabo este experimento una gran cantidad de
| veces.
```

```
...
```

```
|=====
====| 68%
| Con R lo puedes justificar de manera intuitiva, aplicando este experime
| nto muchas veces. En este caso repite el
| experimento 100 000 veces y guárdalo en la variable 'resultado'.
```

```
> resultado<- rbinom(100 000, 7, 0.5)
Error: unexpected numeric constant in "resultado<- rbinom(100 000"
> resultado<- rbinom(100000, 7, 0.5)
```

```
| ¡Todo ese trabajo está rindiendo frutos!
```

```
|=====
=====| 71%
| Como ya te podrás imaginar, 'resultado' contiene el número de caras que
| salieron por cada experimento; en este caso 100
| 000. Ingresa tail(resultado) para conocer el resultado de los últimos 6
| experimentos.
```

```
> tail(resultado)
[1] 3 4 4 5 1 3
```

```
| ¡Muy bien!
```

```
|=====
=====| 75%
| Una vez teniendo los resultados de aplicar este experimento 100 000 vec
| es, puedes encontrar cuántos de esos
| experimentos tuvieron 5 veces o más éxito. Ingresa tail(resultado > 5)
| en la línea de comandos.
```

```
> tail(resultado > 5)
[1] FALSE FALSE FALSE FALSE FALSE FALSE
```

```
| ¡Lo has logrado! ¡Buen trabajo!
```

```
|=====
=====| 79%
| Teniendo este vector ahora puedes encontrar la proporción de verdaderos
contra falsos sacando la media. Ingresar
| mean(resultado > 5) en la línea de comandos.
```

```
> mean(resultado > 5)
[1] 0.06225
```

| ¡Todo ese trabajo está rindiendo frutos!

```
|=====
=====| 82%
| Con esto has encontrado la probabilidad de tener al menos 5 caras al la
nazar 7 veces una moneda justa.
```

...

```
|=====
=====| 86%
| Te preguntará cómo sacaste la media de un vector de valores lógicos; r
ecuerda que esto es posible debido a la
| coerción.
```

...

```
|=====
=====| 89%
| Por último, es posible que desees que tus resultados sean replicables.
La función set.seed() te permite establecer el
| punto de inicio en la generación de números aleatorios. Ingresar ?set.se
ed en la línea de comandos.
```

```
> ?set.seed
```

| ¡Lo estás haciendo muy bien!

```
|=====
=====| 93%
| Como verás, si estableces la misma semilla antes de generar números Ale
atorios, siempre obtendrás los mismos números
| aleatorios.
```

...

```
|=====
=====| 96%
| Ten en cuenta que no siempre es lo que puedas necesitar.
```

...

```
|=====
=====| 100%
| Has concluido la lección. ¿Te gustaría que se le notificará a Coursera
que has completado esta lección?
```

1: si



2: No

Selection: 1

¿Cuál es tu nombre de usuario registrado en Coursera (email)? diego.per  
ez.01@alu.ucm.cl

¿Cuál es tu token de la tarea? nopMIPcU5lP80hqA

El envío de la calificación fue satisfactorio!

| ¡Bien hecho!

| ¡Has alcanzado el fin de esta lección! volviendo al menú principal...