R básico

10-2022

Sección 1

Conociendo R

¿Qué es R?

- Entorno de programación para el análisis estadístico y gráfico de datos
- Software libre
- Sintaxis sencilla e intuitiva
- Enorme comunidad de usuarios (Comprehensive R Archive Network, CRAN)
- ¿Aún tenéis dudas de por qué usarlo? Hay muchas opiniones en la web

¿Qué es RStudio?

En este curso usaremos RStudio-desktop como interfaz gráfica de usuario de R para todos los sistemas operativos

Es un entorno integrado para utilizar y programar con R

Cómo instalar R

Si sois de Windows o Mac

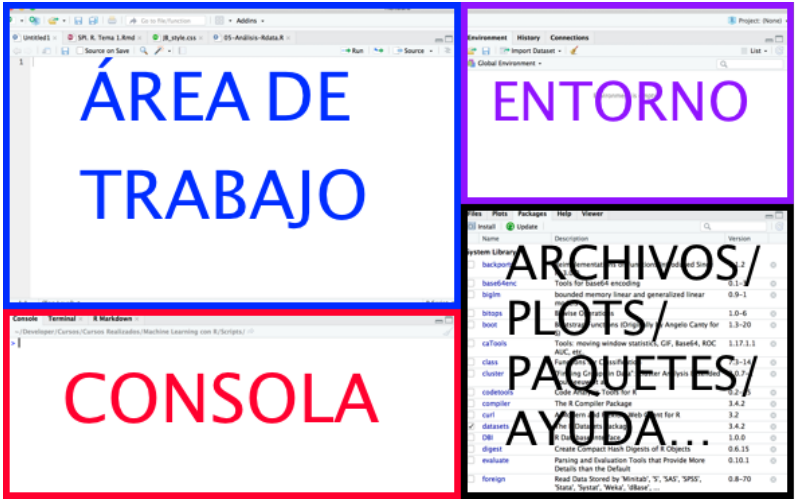
1. Id a CRAN
2. Pulsad sobre el enlace correspondiente a vuestro sistema operativo
3. Seguid las instrucciones de instalación correspondientes

Si trabajáis con Ubuntu o Debian

1. Abrid la terminal, estando conectados a internet
2. Introducid lo siguiente: `sudo aptitude install r-base`

Rstudio

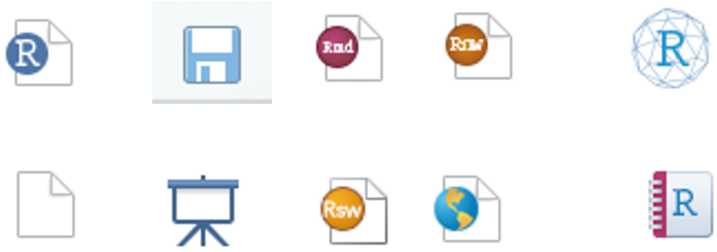
Un editor de `R` y muchas más cosas



Cómo instalar RStudio

1. Obtener RStudio
2. **Solo si utilizáis Linux**, ejecutad en una terminal la siguiente instrucción para completar la instalación: `sudo dpkg -i rstudio-<version>-i386.deb`, donde `version` refiere a la versión concreta que se haya descargado

Trabajando con RStudio



Cómo pedir ayuda

- `help()`: obtener ayuda por consola
- `??...`: obtener ayuda por consola
- Pestaña `Help` de Rstudio
- Cheat Sheet de RStudio y más
- Buscad por la red (stackoverflow, R project...)

Paquetes: cómo instalarlos y cargarlos

Paquete/librería. Un **package** es una librería de funciones y datos que que pueden venir o no instaladas en la carga de R básico.

- `install.packages("nombre_paquete", dep = TRUE)`: instala o actualiza un paquete de R
- `library(nombre_del_paquete)`: carga un paquete ya instalado

Sección 2

Operaciones básicas

Operaciones

Código	Operación
+	Suma
-	Resta
*	Multiplicación
/	División
^	Potencia
%/%	Cociente entero
%%	Resto de división entera

Calculadora básica - Operaciones

Código	Significado
`pi`	$[\pi]$
`Inf`	∞
`NaN`	Indeterminación (Not a Number)
`NA`	Valor desconocido (Not Available)

Calculadora básica - Operaciones

```
2+2
```

```
[1] 4
```

```
77%/%5
```

```
[1] 15
```

```
77%%5
```

```
[1] 2
```

Funciones básicas

Código	Función
`sqrt(x)`	$\sqrt{x}$
`exp(x)`	e^x
`log(x)`	$\ln(x)$
`log10(x)`	$\log_{10}(x)$
`log(x,a)`	$\log_a(x)$
`abs(x)`	$\lvert x \rvert$

Funciones básicas

```
sqrt(9)
```

```
[1] 3
```

```
log(exp(1))
```

```
[1] 1
```

```
log(1000,10)
```

```
[1] 3
```

```
log10(1000)
```

```
[1] 3
```

Combinatoria básica

Código	Operación
`factorial(x)`	x!
`choose(n,m)`	$\binom{n}{m}$

- Número factorial.

  Se define como número factorial de un número entero positivo n como $n! = n \cdot (n-1) \cdots 2 \cdot 1$

- Coeficiente binomial. Se define el coeficiente binomial de n sobre m como

$$\binom{n}{m} = \frac{n!}{m!(n-m)!}$$

Calculadora básica - Combinatoria

```
factorial(5)
```

```
[1] 120
```

```
choose(4,2)
```

```
[1] 6
```

```
factorial(6)
```

```
[1] 720
```

```
factorial(5)*6
```

```
[1] 720
```

Trigonometría en radianes

Código	Función
`sin(x)`	$\sin(x)$
`cos(x)`	$\cos(x)$
`tan(x)`	$\tan(x)$
`asin(x)`	$\arcsin(x)$
`acos(x)`	$\arccos(x)$
`atan(x)`	$\arctan(x)$

Trigonometría en radianes

```
sin(pi/2)
```

```
[1] 1
```

```
cos(pi)
```

```
[1] -1
```

```
tan(0)
```

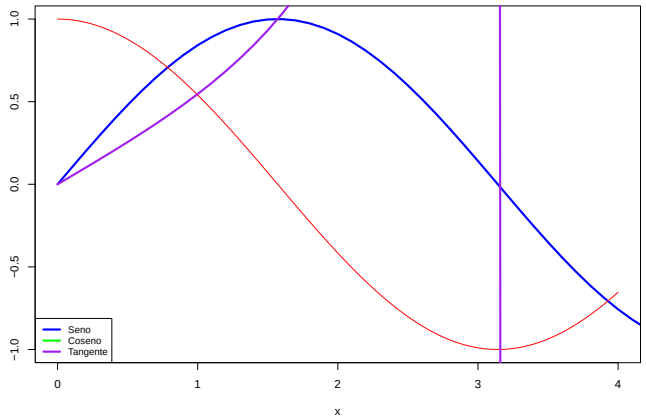
```
[1] 0
```

Un ejemplo de gráficos

```
x = seq(0,2*pi,0.1)
plot(x,sin(x),type="l",col="blue",lwd=3,
     xlab=expression(x), ylab="",
     xlim=c(0,4),cex=0.5)
curve(cos(x),col="red",add=TRUE)
lines(x, tan(x/2), col="purple",lwd=3)
legend("bottomleft",
       col=c("blue","green","purple"),
       legend=c("Seno","Coseno", "Tangente"),
       lwd=3, bty="l",cex=0.8)
```

Un ejemplo de gráficos

. . . en tamaño normal



Números en coma flotante

Código	Función
`print(x,n)`	Muestra las n cifras significativa del número x
`round(x,n)`	Redondea a n cifras significativas un resultado o vector numérico x
`floor(x)`	$\lfloor x \rfloor$, parte entera por defecto de x
`ceiling(x)`	$\lceil x \rceil$, parte entera por exceso de x
`trunc(x)`	Parte entera de x, eliminando la parte decimal

Números en coma flotante

```
print(pi,5)
```

```
[1] 3.1416
```

```
round(pi,5)
```

```
[1] 3.14159
```

```
floor(pi)
```

```
[1] 3
```

```
ceiling(pi)
```

```
[1] 4
```

Variables y funciones

- `nombre_variable = valor`: define una variable con dicho valor
- `nombre_función = function(variable){función}`: define una función

```
a= 8
cubo = function(x){x^3}
cubo(x=a)
```

```
[1] 512
```

```
raiz_cúbica = function(x){x^(1/3)}
raiz_cúbica(a)
```

```
[1] 2
```

```
raiz_cúbica(cubo(x=a))
```

```
[1] 8
```

Sección 3

R Markdown

Introducción

R Markdown. Es un tipo de fichero-programa en el cual podemos intercalar sin problema alguno texto, código y fórmulas matemáticas.

Para la mayor parte de las necesidades de este curso, en lo referente a la creación y composición de este tipo de ficheros, el documento *Markdown Quick Reference* y la chuleta de R Markdown deberían ser suficientes.

Sin embargo, a lo largo de este curso iremos ampliando estos contenidos en algunos temas cuando lo creamos necesario.

Nosotros, en este tema, veremos cómo controlar el comportamiento de los bloques de código (chunks) al compilar el fichero R Markdown y cómo escribir fórmulas matemáticas bien formateadas.

Fórmulas matemáticas

Para escribir fórmulas matemáticas bien formateadas utilizaremos la sintaxis LaTeX

- Para tener ecuaciones o fórmulas en el mismo párrafo, escribimos nuestro código entre dos símbolos de dólar: código
- Si queremos tener ecuaciones o fórmulas centradas en un párrafo aparte, escribimos nuestro código entre dos dobles símbolos de dólar: código

¡Cuidado! Al escribir una fórmula de la forma indicada anteriormente o simplemente texto en R Markdown, los espacios en blanco son completamente ignorados. RStudio solamente añade los espacios en blanco a partir del significado lógico de sus elementos.

Símbolos

Hay muchísimos símbolos matemáticos que puedes escribirse con la sintaxis LaTeX. En el ejemplo anterior ya os hemos mostrado unos pocos. En este tema, nosotros solo veremos los más utilizados.

Para quien quiera ir más allá, aquí os dejamos un documento muy útil con gran cantidad de símbolos de LaTeX.

Símbolos matemáticos - Básico

Significado	Código	Resultado
Suma	`+`	$+$
Resta	`-`	$-$
Producto	`\cdot`	$\cdot$
Producto	`\times`	$\times$
División	`\div`	$\div$
Potencia	`a^{x}`	a^{x}
Subíndice	`a_{i}`	a_{i}

Símbolos matemáticos - Básico

Significado	Código	Resultado
Fracción	`\frac{a}{b}`	$\frac{a}{b}$
Más menos	`\pm`	$\pm$
Raíz n-ésima	`\sqrt[n]{x}`	$\sqrt[n]{x}$
Unión	`cup`	$\cup$
Intersección	`\cap`	$\cap$
OR lógico	`\vee`	$\vee$
AND lógico	`\wedge`	$\wedge$

Símbolos matemáticos - Relaciones

Significado	Código	Resultado
Igual	`=`	$=$
Aproximado	`\approx`	$\approx$
No igual	`\ne`	$\neq$
Mayor que	`>`	$>$
Menor que	`<`	$<$
Mayor o igual que	`\geq`	$\geq$
Menor o igual que	`\leq`	$\leq$

Símbolos matemáticos - Operadores

Significado	Código	Resultado
Sumatorio	`\sum_{i=0}^{n}`	$\sum_{i=0}^{n}$
Productorio	`\prod_{i=0}^{n}`	$\prod_{i=0}^{n}$
Integral	`\int_{a}^{b}`	$\int_{a}^{b}$
Unión (grande)	`\bigcup`	$\bigcup$
Intersección (grande)	`\bigcap`	$\bigcap$
OR lógico (grande)	`\bigvee`	$\bigvee$
AND lógico (grande)	`\bigwedge`	$\bigwedge$

Símbolos matemáticos - Delimitadores

Significado	Código	Resultado
Paréntesis	`()`	$( )$
Corchetes	`[]`	$[ ]$
Llaves	`\{ \}`	$\{ \}$
Diamante	`\langle \rangle`	$\langle \rangle$
Parte entera por defecto	`\lfloor \rfloor`	$\lfloor \rfloor$
Parte entera por exceso	`\lceil \rceil`	$\lceil \rceil$
Espacio en blanco	`hola\ caracola`	*hola caracola*

Símbolos matemáticos - Letras griegas

Significado	Código	Resultado
Alpha	`\alpha`	α
Beta	`\beta`	β
Gamma	`\gamma \Gamma`	γ Γ
Delta	`\delta \Delta`	δ Δ
Epsilon	`\epsilon`	ϵ
Epsilon	`\varepsilon`	ε
Zeta	`\zeta`	ζ

Símbolos matemáticos - Letras griegas

Significado	Código	Resultado
Eta	`\eta`	η
Theta	`\theta \Theta`	θ Θ
Kappa	`\kappa`	κ
Lambda	`\lambda \Lambda`	λ Λ
Mu	`\mu`	μ
Nu	`\nu`	ν
Xi	`\xi \Xi`	ξ Ξ

Símbolos matemáticos - Letras griegas

Significado	Código	Resultado
Pi	`\pi \Pi`	π Π
Rho	`\rho`	ρ
Sigma	`\sigma \Sigma`	σ Σ
Tau	`\tau`	τ
Upsilon	`\upsilon \Upsilon`	υ Υ
Phi	`\phi \Phi`	ϕ Φ
Phi	`\varphi`	φ

Símbolos matemáticos - Letras griegas

Significado	Código	Resultado
Chi	`\chi`	χ
Psi	`\psi \Psi`	ψ Ψ
Omega	`\omega \Omega`	ω Ω

Símbolos matemáticos - Acentos matemáticos

Significado	Código	Resultado
Gorrito	`\hat{x}`	$\hat{x}$
Barra	`\bar{x}`	$\bar{x}$
Punto 1	`\dot{x}`	$\dot{x}$
Punto 2	`\ddot{x}`	$\ddot{x}$
Punto 3	`\dddot{x}`	$\dddot{x}$
Tilde	`\tilde{x}`	$\tilde{x}$
Vector	`\vec{x}`	$\vec{x}$

Símbolos matemáticos - Acentos expansibles

Significado	Código	Resultado
Gorrito	`\widehat{xyz}`	$\widehat{xyz}$
Barra	`\overline{xyz}`	$\overline{xyz}$
Subrallado	`\underline{xyz}`	$\underline{xyz}$
Llave superior	`\overbrace{xyz}`	$\overbrace{xyz}$
Llave inferior	`\underbrace{xyz}`	$\underbrace{xyz}$
Tilde	`\widetilde{xyz}`	$\widetilde{xyz}$
Vector	`\overrightarrow{xyz}`	$\overrightarrow{xyz}$

Símbolos matemáticos - Flechas

Significado	Código	Resultado
Simple	`\leftarrow \rightarrow`	$\leftarrow \rightarrow$
Doble	`\Leftarrow \Rightarrow`	$\Leftarrow \Rightarrow$
Simple larga	`\longleftarrow` `\longrightarrow`	$\longleftarrow \longrightarrow$
Doble larga	`\Longleftarrow` `\Longrightarrow`	$\Longleftarrow \Longrightarrow$
Doble sentido simple	`\leftrightarrow`	$\leftrightarrow$
Doble sentido doble	`\Leftrightarrow`	$\Leftrightarrow$

Símbolos matemáticos - Flechas

Significado	Código	Resultado
Doble sentido larga simple	`\longleftrightarrow`	$\longleftrightarrow$
Doble sentido larga doble	`\Longleftrightarrow`	$\Longleftrightarrow$
Mapea	`\mapsto`	$\mapsto$
Arriba	`\uparrow`	$\uparrow$
Abajo	`\downarrow`	$\downarrow$

Símbolos matemáticos - Funciones

Significado	Código	Resultado
Seno	`\sin`	sin
Coseno	`\cos`	cos
Tangente	`\tan`	tan
Arcoseno	`\arcsin`	arcsin
Arcocoseno	`\arccos`	arccos
Arcotangente	`\arctan`	arctan

Símbolos matemáticos - Funciones

Significado	Código	Resultado
Exponencial	`\exp`	exp
Logaritmo	`\log`	log
Logaritmo neperiano	`\ln`	ln
Máximo	`\max`	max
Mínimo	`\min`	min
Límite	`\lim`	lim

Símbolos matemáticos - Funciones

Significado	Código	Resultado
Supremo	`\sup`	sup
Ínfimo	`\inf`	inf
Determinante	`\det`	det
Argumento	`\arg`	arg

Símbolos matemáticos - Otros

Significado	Código	Resultado
Puntos suspensivos bajos	`\ldots`	$\ldots$
Puntos suspensivos centrados	`\cdots`	$\cdots$
Puntos suspensivos verticales	`\vdots`	$\vdots$
Puntos suspensivos diagonales	`\ddots`	$\ddots$
Cuantificador existencial	`\exists`	$\exists$
Cuantificador universal	`\forall`	$\forall$
Infinito	`\infty`	∞

Símbolos matemáticos - Otros

Significado	Código	Resultado
Aleph	`\aleph`	$\aleph$
Conjunto vacío	`\emptyset`	$\emptyset$
Negación	`\neg`	$\neg$
Barra invertida	`\backslash`	$\backslash$
Dollar	`\$`	$
Porcentaje	`\%`	%
Parcial	`\partial`	∂

Símbolos matemáticos - Tipos de letra

Significado	Código	Resultado
Negrita	`\mathbf{palabra}`	$\mathbf{palabra}$
Negrita	`\boldsymbol{palabra}`	$\boldsymbol{palabra}$
Negrita de pizarra	`\mathbb{NZQRC}`	$\mathbb{NZQRC}$
Caligráfica	`\mathcal{NZQRC}`	$\mathcal{NZQRC}$
Gótica	`\mathfrak{NZQRC}`	$\mathfrak{NZQRC}$

Observaciones

- A la hora de componer en el interior de un párrafo una fracción, existen dos formas: adaptada al tamaño del texto,`$\frac{a}{b}$`, que resulta en $\frac{a}{b}$; o a tamaño real, `$\dfrac{a}{b}$`, que da lugar a $\dfrac{a}{b}$.
- Podemos especificar que los delimitadores se adapten a la altura de la expresión que envuelven utilizando `\left` y `\right`. Observad el cambio en el siguiente ejemplo: `$(\dfrac{a}{b})$` y `$\left(\dfrac{a}{b}\right)$` producen, respectivamente $(\dfrac{a}{b})$ y $\left(\dfrac{a}{b}\right)$.

Matrices

```
$$\begin{matrix} a_{11} & a_{12} & a_{13}\\ a_{21} & a_{22} & a_{23}
\end{matrix}$$
```

$$\begin{matrix} a_{11} & a_{12} & a_{13}\\ a_{21} & a_{22} & a_{23}
\end{matrix}$$

```
$$\begin{pmatrix} a_{11} & a_{12} & a_{13}\\ a_{21} & a_{22} & a_{23}
\end{pmatrix}$$
```

$$\begin{pmatrix} a_{11} & a_{12} & a_{13}\\ a_{21} & a_{22} & a_{23}
\end{pmatrix}$$

Matrices

```
$$\begin{vmatrix} a_{11} & a_{12} & a_{13}\\ a_{21} & a_{22} & a_{23}
\end{vmatrix}$$
```

$$\begin{vmatrix} a_{11} & a_{12} & a_{13}\\ a_{21} & a_{22} & a_{23}
\end{vmatrix}$$

```
$$\begin{bmatrix} a_{11} & a_{12} & a_{13}\\ a_{21} & a_{22} & a_{23}
\end{bmatrix}$$
```

$$\begin{bmatrix} a_{11} & a_{12} & a_{13}\\ a_{21} & a_{22} & a_{23}
\end{bmatrix}$$

Matrices

```
$$\begin{Bmatrix} a_{11} & a_{12} & a_{13}\\ a_{21} & a_{22} & a_{23}
\end{Bmatrix}$$
```

$$\begin{Bmatrix} a_{11} & a_{12} & a_{13}\\ a_{21} & a_{22} & a_{23}
\end{Bmatrix}$$

```
$$\begin{Vmatrix} a_{11} & a_{12} & a_{13}\\ a_{21} & a_{22} & a_{23}
\end{Vmatrix}$$
```

$$\begin{Vmatrix} a_{11} & a_{12} & a_{13}\\ a_{21} & a_{22} & a_{23}
\end{Vmatrix}$$

Sistema de ecuaciones

`\begin{array}{ll}\end{array}` nos produce una tabla alineada a la izquierda. El hecho de introducir el código `\left. \right.` hace que el delimitador respectivo no aparezca.

Sistema de ecuaciones

```
$$\left.\begin{array}{ll} ax+by=& c\\ ex-fy=& g \end{array}\right\}$$
```

$$\left.\begin{array}{ll} ax+by=& c\\ ex-fy=& g \end{array}\right\}$$

```
$$|x|=\left\{\begin{array}{ll} -x & \text{si }x\leq 0\\ x & \text{si }x\geq
0 \end{array}\right. $$
```

$$|x|=\left\{\begin{array}{ll} -x & \text{si } x\leq 0\\ x & \text{si } x\geq 0 \end{array}\right.$$

Sistema de ecuaciones

La función de LaTeX `\text{}` nos permite introducir texto en fórmulas matemáticas.

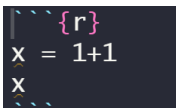
Sección 4

Parámetros de los chuncks de R

Chunks de R

Chunk. Bloque de código.

Los bloques de código de R dentro de un documento R Markdown se indican de la manera siguiente



que resulta en

```r
x = 1+1
x
```

```
[1] 2
```

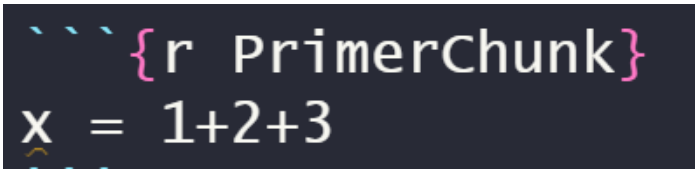
Chunks de R

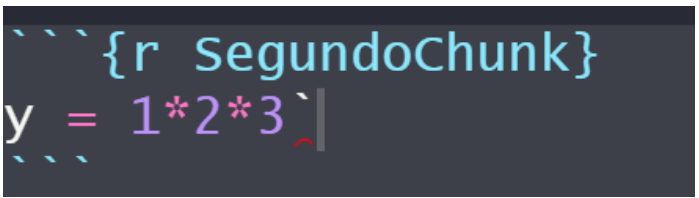
Hay diversas opciones de crear un bloque de código de R:

- Ir al menú desplegable de “Chunks” y seleccionar el de R
- Introducir manualmente
- Alt + Command + I (para Mac) o Alt + Control + I (para Windows)

Chunks de R

A los chunks se les puede poner etiqueta, para así localizarlos de manera más fácil. Por ejemplo





Parámetros de los chunks

La parte entre llaves también puede contener diversos parámetros, separados por comas entre ellos y separados de la etiqueta (o de r, si hemos decidido no poner ninguna).

Estos parámetros determinan el comportamiento del bloque al compilar el documento pulsando el botón `Knit` situado en la barra superior del área de trabajo.

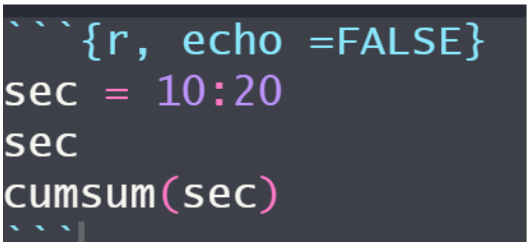
Parámetros de los chunks

Código	Significado
`echo`	Si lo igualamos a `TRUE`, que es el valor por defecto, estaremos diciendo que queremos que se muestre el código fuente del chunk. En cambio, igualado a `FALSE`, no se mostrará
`eval`	Si lo igualamos a `TRUE`, que es el valor por defecto, estaremos diciendo que queremos que se evalúe el código. En cambio, igualado a `FALSE`, no se evaluará

Parámetros de los chunks

Código	Significado
`message`	Nos permite indicar si queremos que se muestren los mensajes que R produce al ejecutar código. Igualado a `TRUE` se muestran, igualado a `FALSE` no
`warning`	Nos permite indicar si queremos que se muestren los mensajes de advertencia que producen algunas funciones al ejecutarse. Igualado a `TRUE` se muestran, igualado a `FALSE` no

Parámetros de los chunks



No aparece el código solo la salida

```
[1] 10 11 12 13 14 15 16 17 18 19 20
```

```
[1]  10  21  33  46  60  75  91 108 126 145 165
```

Parámetros de los chunks

```
```{r, echo=TRUE, message = TRUE}
library(car)
head(cars,3)
```
```

```
library(car)
```

```
Loading required package: carData
```

```
head(cars,3)
```

```
  speed dist
1     4    2
2     4   10
3     7    4
```

Parámetros de los chunks

````
```{r, echo = TRUE, message = FALSE, comment = NA}
library(car)
head(cars,3)
```
````

```
library(car)
head(cars,3)
```

```
  speed dist
1     4    2
2     4   10
3     7    4
```

Fijaos que `comment=NA` evita que aparezcan los ##

Parámetros de los chunks

Significado	Código	Resultado
`results`	`markup`	Valor por defecto. Nos muestra los resultados en el documento final línea a línea, encabezados por ##
`results`	`hide`	No se nos muestra el resultado en el documento final
`results`	`asis`	Nos devuelve los resultados línea a línea de manera literal en el documento final y el programa con el que se abre el documento final los interpreta como texto y formatea adecuadamente

Parámetros de los chunks

```
```{r, echo=TRUE, results='markup'}
sec = 10:20
sec
cumsum(sec)
```
```

```
sec = 10:20
sec
```

```
[1] 10 11 12 13 14 15 16 17 18 19 20
```

```
cumsum(sec)
```

```
[1]  10  21  33  46  60  75  91 108 126 145 165
```

Parámetros de los chunks

```
```{r, echo=TRUE, results='hide'}
sec = 10:20
sec
cumsum(sec)
```
```

```
sec = 10:20
sec
cumsum(sec)
```

Parámetros de los chunks

````
```{r chunk_ex, echo=TRUE, results='asis'}
sec = 10:20
sec
cumsum(sec)
```
````

```
sec = 10:20
sec
```

[1] 10 11 12 13 14 15 16 17 18 19 20

```
cumsum(sec)
```

[1] 10 21 33 46 60 75 91 108 126 145 165

Parámetros de los chunks

````
```{r una_chunk, echo=TRUE, results='hold'}
sec = 10:20
sec
cumsum(sec)
```
````

```r
sec = 10:20
sec
cumsum(sec)
```

```
 [1] 10 11 12 13 14 15 16 17 18 19 20
 [1]  10  21  33  46  60  75  91 108 126 145 165
```

Sección 5

Estructuras de datos

Tipos de datos en R, vectores

Un **vector** es una secuencia ordenada de datos. `R` dispone de muchos tipos de datos, por ejemplo:

- `logical`: lógicos (`TRUE` o `FALSE`)
- `integer`: números enteros, $\mathbb{Z}$
- `numeric`: números reales, $\mathbb{R}$
- `complex`: números complejos, $\mathbb{C}$
- `character`: palabras

En los vectores de `R`, todos sus objetos han de ser del mismo tipo: todos números, todos palabras, etc. Cuando queramos usar vectores formados por objetos de diferentes tipos, tendremos que usar **listas generalizadas**, `lists` que veremos al final del tema.

Básico

- `c()`: para definir un vector
- `scan()`: para definir un vector
- `fix(x)`: para modificar visualmente el vector x
- `rep(a,n)`: para definir un vector constante que contiene el dato a repetido n veces

```
c(1,2,3)
```

```
[1] 1 2 3
```

```
rep("Mates",7)
```

```
[1] "Mates" "Mates" "Mates" "Mates" "Mates" "Mates" "Mates"
```

Función scan()

Ejemplo

Vamos a crear un vector que contenga 3 copias de 1 9 9 8 0 7 2 6 con la función scan:

```
> scan()
1: 1 9 9 8 0 7 2 6
9: 1 9 9 8 0 7 2 6
17: 1 9 9 8 0 7 2 6
25:
Read 24 items
 [1] 1 9 9 8 0 7 2 6 1 9 9 8 0 7 2 6 1 9 9 8 0 7 2 6
>
```

Básico

Ejercicio

1. Repite tu año de nacimiento 10 veces
2. Crea el vector que tenga como entradas $16, 0, 1, 20, 1, 7, 88, 5, 1, 9$, llámalo vec y modifica la cuarta entrada con la función fix()

Progresiones y Secuencias

Una progresión aritmética es una sucesión de números tales que la **diferencia**, d, de cualquier par de términos sucesivos de la secuencia es constante.

$$a_n = a_1 + (n-1) \cdot d$$

- `seq(a,b,by=d)`: para generar una progresión aritmética de diferencia d que empieza en a hasta llegar a b
- `seq(a,b, length.out=n)`: define progresión aritmética de longitud n que va de a a b con diferencia d. Por tanto $d = (b-a)/(n-1)$
- `seq(a,by=d, length.out=n)`: define la progresión aritmética de longitud n y diferencia d que empieza en a
- `a:b`: define la secuencia de números **enteros** ($\mathbb{Z}$) consecutivos entre dos números a y b

Secuencias

Ejercicio

- Imprimid los números del 1 al 20
- Imprimid los 20 primeros números pares
- Imprimid 30 números equidistantes entre el 17 y el 98, mostrando solo 4 cifras significativas

Funciones

Cuando queremos aplicar una función a cada uno de los elementos de un vector de datos, la función `sapply` nos ahorra tener que programar con bucles en `R`:

- `sapply(nombre_de_vector,FUN=nombre_de_función)`: para aplicar dicha función a todos los elementos del vector
- `sqrt(x)`: calcula un nuevo vector con las raíces cuadradas de cada uno de los elementos del vector x

Funciones

Dado un vector de datos x podemos calcular muchas medidas estadísticas acerca del mismo:

- `length(x)`: calcula la longitud del vector x
- `max(x)`: calcula el máximo del vector x
- `min(x)`: calcula el mínimo del vector x
- `sum(x)`: calcula la suma de las entradas del vector x
- `prod(x)`: calcula el producto de las entradas del vector x

Funciones

- `mean(x)`: calcula la media aritmética de las entradas del vector x
- `diff(x)`: calcula el vector formado por las diferencias sucesivas entre entradas del vector original x
- `cumsum(x)`: calcula el vector formado por las sumas acumuladas de las entradas del vector original x
  - Permite definir sucesiones descritas mediante sumatorios
  - Cada entrada de `cumsum(x)` es la suma de las entradas de x hasta su posición

Funciones

```
cuadrado = function(x){x^2}
v = c(1,2,3,4,5,6)
sapply(v, FUN = cuadrado)
```

```
[1]  1  4  9 16 25 36
```

```
mean(v)
```

```
[1] 3.5
```

```
cumsum(v)
```

```
[1]  1  3  6 10 15 21
```

Orden

- `sort(x)`: ordena el vector en orden natural de los objetos que lo forman: el orden numérico creciente, orden alfabético...
- `rev(x)`: invierte el orden de los elementos del vector x

```
v = c(1,7,5,2,4,6,3)
sort(v)
```

```
[1] 1 2 3 4 5 6 7
```

```
rev(v)
```

```
[1] 3 6 4 2 5 7 1
```

Orden

Ejercicio

- Combinad las dos funciones anteriores, `sort` y `rev` para crear una función que dado un vector x os lo devuelva ordenado en orden decreciente.
- Razonad si aplicar primero `sort` y luego `rev` a un vector x daría en general el mismo resultado que aplicar primero `rev` y luego `sort`.
- Investigad la documentación de la función `sort` (recordad que podéis usar la sintaxis `?sort` en la consola) para leer si cambiando algún argumento de la misma podéis obtener el mismo resultado que habéis programado en el primer ejercicio.

Subvectores

- `vector[i]`: da la i-ésima entrada del vector
  - Los índices en R empiezan en 1
  - `vector[length(vector)]`: nos da la última entrada del vector
  - `vector[a:b]`: si a y b son dos números naturales, nos da el subvector con las entradas del vector original que van de la posición a-ésima hasta la b-ésima.
  - `vector[-i]`: si i es un número, este subvector está formado por todas las entradas del vector original menos la entrada i-ésima. Si i resulta ser un vector, entonces es un vector de índices y crea un nuevo vector con las entradas del vector original,cuyos índices pertenecen a i
  - `vector[-x]`: si x es un vector (de índices), entonces este es el complementario de vector[x]

Subvectores

- También podemos utilizar operadores lógicos:
  - `==`: $=$
  - `!=`: $\neq$
  - `>=`: $\geq$
  - `<=`: $\leq$
  - `<`: $<$
  - `>`: $>$
  - `!`: NO lógico
  - `&`: Y lógico
  - `|`: O lógico

Subvectores

```
v = c(14,5,6,19,32,0,8)
v[2]
```

```
[1] 5
```

```
v[-c(3,5)]
```

```
[1] 14  5 19  0  8
```

```
v[v != 19 & v>15]
```

```
[1] 32
```

Condicionales

- `which(x cumple condición)`: para obtener los índices de las entradas del vector x que satisfacen la condición dada
- `which.min(x)`: nos da la primera posición en la que el vector x toma su valor mínimo
- `which(x==min(x))`: da todas las posiciones en las que el vector x toma sus valores mínimos
- `which.max(x)`: nos da la primera posición en la que el vector x toma su valor máximo
- `which(x==max(x))`: da todas las posiciones en las que el vector x toma sus valores máximos