# Introducción a R

Rodrigo Santamaría

# Introducción a R

¿Qué es R?
Variables
Funciones
Control de flujo
Manejo de gráficas

# Introducción a R

¿Qué es R?

Usos de R

BioConductor

Descarga e instalación

Organización

Comentarios y ayuda

Variables

Funciones

Control de flujo

Manejo de gráficas

# ¿Qué es R?

- R es un lenguaje de programación para estadísticos
  - Un lenguaje de programación es un modo de comunicarnos con los ordenadores para que hagan cálculos
- Peculiaridades
  - Es un lenguaje interpretado:
    - tan pronto como escribimos una orden obtenemos su resultado
  - Es un lenguaje vectorial:
    - las operaciones se realizan a la vez para todo un vector o matriz

# ¿Qué es R?

- ¿Por qué usar R?
  - Es gratis
  - Tiene una extensión, Bioconductor
    - Con multitud de utilidades en bioinformática, que crece cada día
    - Muy aceptada por la comunidad científica
- Otras opciones
  - Matlab:
    - De pago, pero con un paquete en bioinformática muy completo
    - Útil para hacer informes, pero menos rápidamente actualizado
  - Perl
    - Muy útil para análisis de secuencias (BioPerl)

# Usos de R

## Usuario

- Utiliza las funciones ya desarrolladas por otros
- Destrezas
  - Conocer las diferentes métodos disponibles para una tarea
  - Conocer las diferentes opciones de cada método
  - Manejar los formato de E/S → ensamblar métodos

## Programador

- Crea nuevas funciones para que otros las puedan utilizar
- Destrezas
  - Conocimientos avanzados de programación en R
  - Conocimiento profundo de los métodos a desarrollar
  - A veces, conocimiento de otros lenguajes (C, Java)

# Usos de R

- Nos quedaremos a nivel de usuario
  - Con mínimas intrusiones en el nivel de programador
- ¿Por qué usar R con todas las herramientas chulas que hay a nivel de usuario para bioinformática?
  - A veces las herramientas chulas no hacen exactamente lo que queremos
    - O no encajan bien en nuestro flujo de trabajo (formatos, etc.)
  - Generalmente esas herramientas aparecen más tarde que los paquetes en R/BioConductor

# BioConductor

- Extensión de R con paquetes de utilidades bioinformáticas
  - Análisis de microarrays
    - Probablemente la parte en que BioConductor es más fuerte
  - Anotaciones
    - Otro de los fuertes de BioConductor, con acceso a multitud de bases de datos con información genética
  - Análisis de secuencias
    - Permite manejar y explorar diferentes formatos de secuencias, pero es limitado a nivel de análisis
  - Experimentos de alto rendimiento
    - Todavía un campo muy nuevo pero ya hay bastantes funcionalidades

# BioConductor: paquetes que usaremos

- Análisis de secuencias
  - `Biostrings`, `annotate`
- Análisis de microarrays
  - `ArrayExpress`, `GEOquery`, `limma`
- Anotaciones
  - `biomaRt`, `GO`, `KEGG`
- High Throughtput Sequencing (HTS)
  - `ShortRead`, `rtracklayer`, `ArrayExpressHTS`

# Descarga e instalación

- Descarga de R
  - http://cran.r-project.org/
- Instalación de paquetes de BioConductor
  - `source("http://bioconductor.org/biocLite.R")`
  - `biocLite() #instalamos los paquetes base`
  - `biocLite("nombrePaquete") #paquetes particulares`

# Organización

- R se organiza en paquetes
  - Contenedores de funciones y (opcionalmente) datos
- Para cargar un paquete: `library(nombrePaquete)`
  - Debe estar previamente instalado
  - Si no se carga no se pueden usar sus métodos
- Por ejemplo, si queremos hacer un alineamiento de pares de secuencias mediante la función `pairwiseAlignment`
  - Tendremos que instalar y cargar el paquete `Biostrings` que contiene dicha función

# Comentarios y ayuda

```
> #esto es un comentario de una línea (empiezan por #)
> #los comentarios son ignorados por R
> #pero son MUY IMPORTANTES para explicar el código

> ?data.frame    #despliegua la ayuda de una función
```

data.frame {base} R Documentation

**Data Frames**

**Description**

This function creates data frames, tightly coupled collections of variables which share many of the properties of matrices and of lists, used as the fundamental data structure by most of R's modeling software.

**Usage**

```
data.frame(..., row.names = NULL, check.rows = FALSE,
           check.names = TRUE,
           stringsAsFactors = default.stringsAsFactors())

default.stringsAsFactors()
```

**Arguments**

| | |
|---|---|
| ... | these arguments are of either the form `value` or `tag = value`. Component names are created based on the tag (if present) or the deparsed argument itself. |
| row.names | NULL or a single integer or character string specifying a column to be used as row names, or a character or integer vector giving the row names for the data |

# Comentarios y ayuda

```
> #esto es un comentario de una línea (empiezan por #)
> #los comentarios son ignorados por R
➢#pero son MUY IMPORTANTES para explicar el código

> ?data.frame   #despliegua la a
```

data.frame {base} R Documentation

**Data Frames**

**Description**

This function creates data frames, tightly coupled collections of variables which share many of the properties of matrices and of lists, used as the fundamental data structure by most of R's modeling software.

**Usage**

```
data.frame(..., row.names = NULL, check.rows = FALSE,
           check.names = TRUE,
           stringsAsFactors = default.stringsAsFactors())

default.stringsAsFactors()
```

**Arguments**

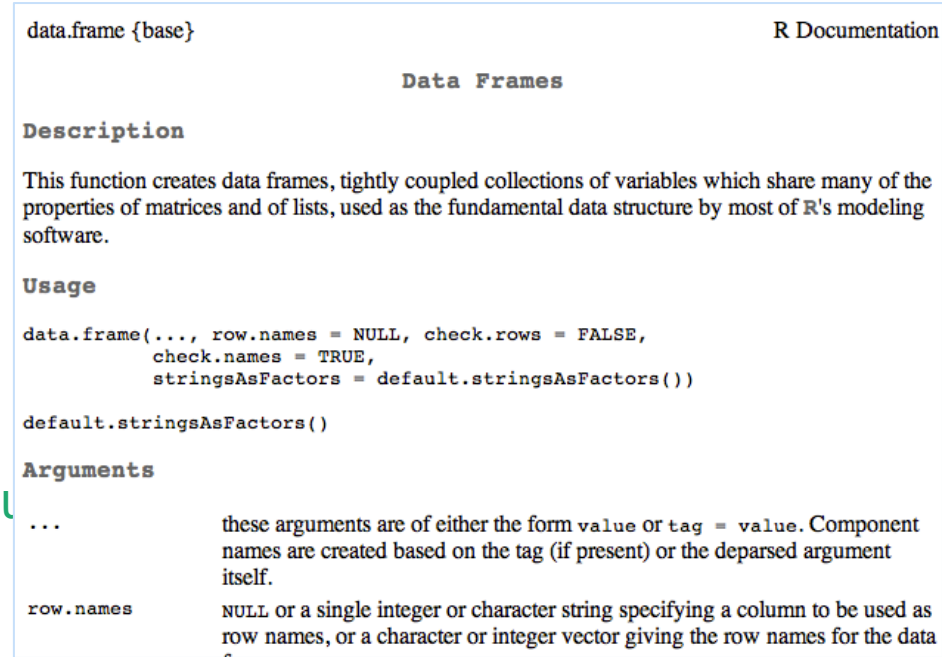
| | |
|---|---|
| ... | these arguments are of either the form `value` or `tag = value`. Component names are created based on the tag (if present) or the deparsed argument itself. |
| row.names | NULL or a single integer or character string specifying a column to be used as row names, or a character or integer vector giving the row names for the data |

# Comentarios y ayuda

```
> #esto es un comentario de una línea (empiezan por #)
> #los comentarios son ignorados por R
```

- #pero son MUY IMPORTANTES para explicar el código

```
> ?data.frame  #despliegu
```



data.frame {base} R Documentation

**Data Frames**

**Description**

This function creates data frames, tightly coupled collections of variables which share many of the properties of matrices and of lists, used as the fundamental data structure by most of R's modeling software.

**Usage**

```
data.frame(..., row.names = NULL, check.rows = FALSE,
           check.names = TRUE,
           stringsAsFactors = default.stringsAsFactors())

default.stringsAsFactors()
```

**Arguments**

| | |
|---|---|
| ... | these arguments are of either the form `value` or `tag = value`. Component names are created based on the tag (if present) or the deparsed argument itself. |
| row.names | NULL or a single integer or character string specifying a column to be used as row names, or a character or integer vector giving the row names for the data |

# Viñetas (vignettes)

- La mayoría de los paquetes, aparte de la ayuda por línea de comandos, incluyen documentación en PDF (viñetas)
- Estas viñetas incluyen la ayuda en línea y a menudo ejemplos, guías o tutoriales sobre su uso

```
> vignette("limma") #Abre la viñeta del paquete limma
> vignette("ArrayExpress")
> browseVignettes() #Lista todas la viñetas disponibles
```

# Introducción a R

¿Qué es R?

Variables

- Vectores
- Matrices
- data.frame
- Clases

Funciones

Control de flujo

Manejo de gráficas

# Variables

- Una variable es un elemento que puede tomar valores
  - Las variables en R no tienen tipos ni se tienen que declarar
- Una variable de nombre seq con valor “AGGTCCGT”
  - `seq <- “AGGTCCGT”`
- Una variable num que vale 3
  - `num = 3`
- `<-` y = son equivalentes
  - Asignan un valor a una variable

# Vectores

- Conjunto de variables del mismo tipo

```
> vector <- c(1,3,4,5,66,77) # vector con 6 números
> cadena <- c("A", "C", "C", "G", "A", "T") #y con 6 letras
> length(cadena) #longitud de la cadena
[1] 6
> cadena[3]  #acceso a elementos particulares
[1] "C"
> vector[2:4] #o a intervalo de elementos
[1] 3 4 5
```

# Matrices

- Conjunto bidimensional de variables del mismo tipo

```
> mat <- matrix(c(2,0,3,4), 2,2)
> mat
     [,1] [,2]
[1,]    2    3
[2,]    0    4
> mat[,2] #acceso a una columna
[1] 3 4
> mat[1,1] #acceso a un elemento
[1] 2
> mat[1,] #acceso a una fila
[1] 2 3
```

# Matrices

```
> colnames(mat)=c("col1", "col2") #pone nombres a cols
> mat
     col1 col2
[1,]    2    3
[2,]    0    4
> rownames(mat)=c("row1", "row2")
> mat
     col1 col2
row1    2    3
row2    0    4
> rownames(mat) #o nos da sus nombres
[1] "row1" "row2”
> dim(mat)   #nos da un vector con las dimensiones
[1] 2 2
```

# Matrices

```
> cbind(mat, c(9,9)) #añade un vector como columna
     col1 col2
row1    2    3 9
row2    0    4 9
> rbind(mat, c(7,7)) #o una fila
     col1 col2
row1    2    3
row2    0    4
        7    7
> mat<-rbind(mat, c(7,7)) #así almacenamos la nueva matriz
> mat
     col1 col2
row1    2    3
row2    0    4
        7    7
```

# data.frame

- Matriz donde cada columna puede ser de un tipo

```
> df=data.frame(sexo=c("H", "M", "M"), edad=c(23, 44, 18))
> df
  sexo edad
1    H   23
2    M   44
3    M   18
> colnames(df)   #los nombres de columnas que pusimos
[1] "sexo" "edad"
> rownames(df)   #pone automáticamente nombres a las filas
[1] "1" "2" "3"
```

# lectura de ficheros

- read.table
  - file → ruta al fichero
  - sep → separador entre columnas
  - para más opciones: ?read.table

```
> tabla=read.table(file="/Users/rodri/Documents/workspace/
sybaris/lps/LPS_rep1dig.txt", sep="\t")
> class(tabla)  #las tablas se almacenan en data.frames
[1] "data.frame"
> dim(tabla)    #que podemos manejar como una matriz
[1] 24628    25
> tabla[1,1]
[1] Mus musculus/biomaRt.entrezgene
```

# Clases

- Tipos especiales de variables
  - Pueden contener otras variables
  - Hay métodos que sólo funcionan sobre determinadas clases
- Por ejemplo, la clase ExpressionSet de Biobase
  - `new("ExpressionSet", phenoData = new("AnnotatedDataFrame"), featureData = new("AnnotatedDataFrame"), experimentData = new("MIAME"), annotation = character(0), protocolData = phenoData[,integer(0)], exprs = new("matrix"))`
  - Contiene dos variables de tipo AnnotatedDataFrame, una de tipo MIAME, una matriz, etc.

# Clases

```
#Esta función devuelve un objeto de tipo ExpressionSet
> eset=ArrayExpress("E-TABM-25”)
> eset
AffyBatch object
size of arrays=640x640 features (40 kb)
cdf=HG_U95Av2 (12625 affyids)
number of samples=18
number of genes=12625
annotation=hgu95av2
notes=E-TABM-25
    E-TABM-25
    c("organism_part", "age")
    c("physiological_process_design",
"organism_status_design", "transcription profiling by
array")
```

# Clases

```
#Podemos acceder a sus variables con métodos propios de la
clase, o en ciertos casos con el operador @

> eset@annotation
[1] "hgu95av2"
> annotation(eset)
[1] "hgu95av2"
> eset@protocolData
An object of class "AnnotatedDataFrame"
  sampleNames: cp7.CEL cp4.CEL ... cc1.CEL (18 total)
  varLabels: ScanDate
  varMetadata: labelDescription
> dim(exprs(eset))
[1] 409600     18
```

# Introducción a R

¿Qué es R?

Variables

Funciones

- Definición
- Operaciones aritmético-lógicas
- Operaciones estadísticas
- Operaciones con cadenas

Control de flujo

Manejo de gráficas

# Funciones

- Operaciones sobre las variables
  - También llamada métodos

```
retorno <- funcion(argumento1, argumento2, ..., argumentoN)
```

  - argumento1…N son variables que necesita la función
    - También llamados variables de entrada, parámetros.
  - retorno es la variable que retorna la función como resultado de su operación (también llamada variable de salida)

```
> dim(mat) #la función dim() acepta una matriz como argumento
           #y devuelve un vector son su dimensión
[1] 3 2
```

# Operaciones aritméticas

```
> 2+2 #análogamente -, * y /
[1] 4
> vector
[1]  1  3  4  5 66 77
> vector-1
[1]  0  2  3  4 65 76
> vector/3
[1]  0.33333  1.00000  1.33333  1.66667 22.00000 25.66667
> vector^2   #elevar al cuadrado (u otro valor)
[1]    1    9   16   25 4356 5929
> log(55)    #logaritmo neperiano (base e). log2, log10…
[1] 4.007333
> exp(3)     #potencia exponencial (~ 2.7172^3)
[1] 20.08554
```

# Operaciones lógicas

```
> vector
[1]  1  3  4  5 66 77
> vector == 5 #dice qué elementos cumplen (TRUE) la igualdad
[1] FALSE FALSE FALSE  TRUE FALSE FALSE
> vector <= 5 #o desigualdad (también <, >, >=, !=)
[1]  TRUE  TRUE  TRUE  TRUE FALSE FALSE
> vector != 5
[1]  TRUE  TRUE  TRUE FALSE  TRUE  TRUE
> vector > 5
[1] FALSE FALSE FALSE FALSE  TRUE  TRUE
> which(vector==5)  #da las posiciones que cumplen la comp.
[1] 4
> which(vector < 5)
[1] 1 2 3
```

# Operaciones de conjuntos

- Los vectores se pueden tratar como conjuntos

```
> set1=c(1,2,3,4,5,11,12)
> set2=c(0,0,11,12,8,9)
> unique(set2)      #eliminar repetidos
[1]  0 11 12  8  9
> union(set1, set2)
 [1]  1  2  3  4  5 11 12  0  8  9
> intersect(set1, set2)
[1] 11 12
> set1 %in% set2 #qué elementos de set1 están en set2
[1] FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE
> which(set1 %in% set2)#sus posiciones en set1
[1] 6 7
```

# Operaciones estadísticas

```
> vector=c(0,0,1,4,8,9)
> mean(vector)   #media
[1] 3.666667
> median(vector) #mediana
[1] 2.5
> sd(vector)     #desviación típica
[1] 4.033196
> sum(vector)    #sumatorio
[1] 22
> prod(vector)   #producto
[1] 0
```

# Operaciones estadísticas

```
> mat=matrix(c(0,1,3,0,9,7),2,3)
> mat
     [,1] [,2] [,3]
[1,]    0    3    9
[2,]    1    0    7
> mean(mat)      #media de todos los valores
[1] 3.333333
> rowMeans(mat)  #medias de las filas
[1] 4.000000 2.666667
> colMeans(mat)  #medias de las columnas
[1] 0.5 1.5 8.0
> sum(mat)       #suma de todos los valores
[1] 20
> rowSums(mat)   #suma de las filas
[1] 12  8
> colSums(mat)   #suma de las columnas
[1]  1  3 16
```

# Distribuciones

- El paquete `stats` tienen funciones para generar valores aleatorios que sigan una determinada distribución
  - Usa `?Distributions` para verlas todas
  - La más usada es `rnorm(num, mean, sd)`
    - Genera `num` números que siguen una distribución normal centrada en `mean` y con desviación `sd`

```
> rnorm(10)    #Por defecto centrada en 0 con desviación 1
 [1]  0.3742577  1.7460456  0.6507397 -1.3333722  0.2525126
1.7423761
 [7]  0.7139431  1.7796442 -0.1328050 -1.7788405
> rnorm(10, 100,10) #Centrada en 100 con desviación 10
 [1]  87.79390 101.73924  97.12256  94.64889  98.10348 107.73062
96.55949
 [8] 100.98834  98.77138  84.67858
```

# Operaciones con cadenas

```
> paste("V", c(1:5)) #Genera un vector pegando elementos
[1] "V 1" "V 2" "V 3" "V 4" "V 5"
> paste("V", c(1:5), sep="") #Podemos indicar con qué “pega”
[1] "V1" "V2" "V3" "V4" "V5"
> paste(c(1:5), c(9:14), sep="-")
[1] "1-9"  "2-10" "3-11" "4-12" "5-13" "1-14"
```

```
> strsplit("El perro de San Roque no tiene rabo", " ")
[[1]] #Crea un vector cortando elementos por un carácter
[1] "El"    "perro" "de"    "San"   "Roque" "no"
"tiene" "rabo"

> strsplit("El perro de San Roque no tiene rabo", "o")
[[1]]
[1] "El perr"    " de San R"  "que n"      " tiene rab"
```

# Operaciones con cadenas (grep)

```
grep(pattern="Al", x=c("Alicia", "Alba", "Carlos", "Alejandro"))
[1] 1 2 4
```

- grep retorna los índices de los elementos de un vector de cadenas (x) que contienen el patrón (pattern)
- Este patrón es una *expresión regular*, y puede contener caracteres especiales
  - * - el carácter anterior aparece cero o más veces
  - ? – el carácter anterior aparece cero o una vez
  - + - el carácter anterior aparece una o más veces
  - . – un carácter cualquiera
  - Más en ?grep

# grep

```
> nombres=c("Alicia", "Alba", "Manuel", "Carlos", "Alejandro")
> grep("[Aa]", nombres)    #contener una a o una A
[1] 1 2 3 4 5
> grep("A", nombres)       #contener una A
[1] 1 2 5
> grep("^A", nombres)      #comenzar por una A
[1] 1 2 5
> grep("o$", nombres)      #terminar por una o
[1] 5
> grep("A..a", nombres)    #tener una A, luego 2 letras y una a
[1] 2
> grep("A.*a", nombres)  #tener una A, luego cualquier cosa y a
[1] 1 2 5
> grep("A.*a$", nombres)
       #tener una A, luego cualquier cosa y terminar con una a
[1] 1 2
```

# Introducción a R

¿Qué es R?

Variables

Funciones

Control de flujo

- Comparaciones
- Condiciones
- Bucles

Manejo de gráficas

# Control de flujo

- Funciones especiales para determinar el orden en el que se ejecutan las órdenes del programa
  - **Condiciones**: determinan si una orden se ejecuta o no
    - Si X haz Y, si no haz Z
  - **Bucles**: determinan cuántas veces se ejecuta una orden
    - Mientras X haz Y
    - Hasta que X haz Y
- Suelen depender de **comparaciones**
  - Si/mientras X (igual/distinto/mayor) que Y haz Z

# Comparaciones

- Devuelven verdadero o falso dependiendo de si la comparación se cumple o no:
  - x igual a 3? → `x==3`
  - x distinto de y? → `x!=y`
  - 3 mayor que 4? → `3>4` (falso siempre)
  - x menor o igual que 100? → `x<=100`

# Condiciones

```
> vector=c(1,2,3)
> if(vector[2]==2) vector[3]=99
> vector
[1]  1  2 99
> if(sum(vector)<100)  vector[1]=0
> vector
[1]  1  2 99
```

```
> if(mean(vector)>50)
+   { #llaves si +1 orden
+   mayor=TRUE
+   vector=0
+   }
> else  #~ "si no"
+   {
+   mayor=FALSE
+   vector=189
+   }
> mayor
[1] FALSE
> vector
[1] 189
```

# Bucles

```
> vector=c(1,2,3)
> vector2=c()
> for(i in 1:length(vector))
+   {
+   vector2=c(vector2,vector[i]*i)
+   }
> vector2
[1] 1 4 9
```

- Utilizan una variable como contador
  - en este caso la hemos llamado i
- El contador va tomando todos los valores de un vector (en este caso 1,2,3)
- Para cada uno de esos valores, se ejecutan las órdenes entre llaves

# sapply

- Similar a un bucle, pero más compacto
  - Retorna un vector con los valores de la última orden ejecutada para cada iteración

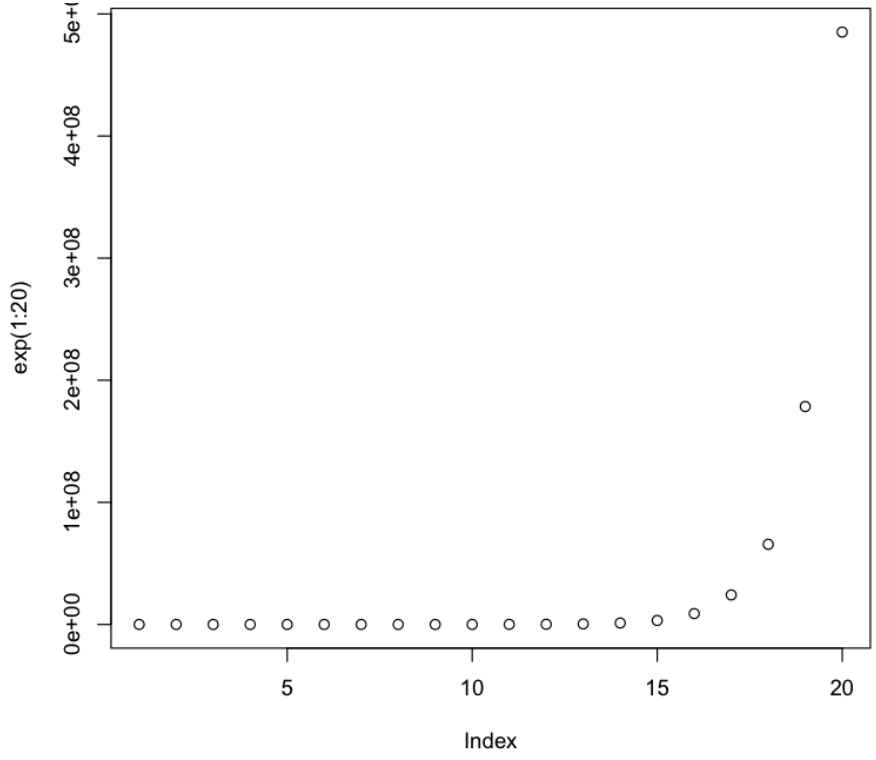
```
sapply(vectorArecorrer, function(iterador){órdenes})

sapply(1:length(vector), function(i){i*vector[i]})

sapply(vector, function(x){x^3})

sapply(vector, function(x){x=x*2; x^3})
```

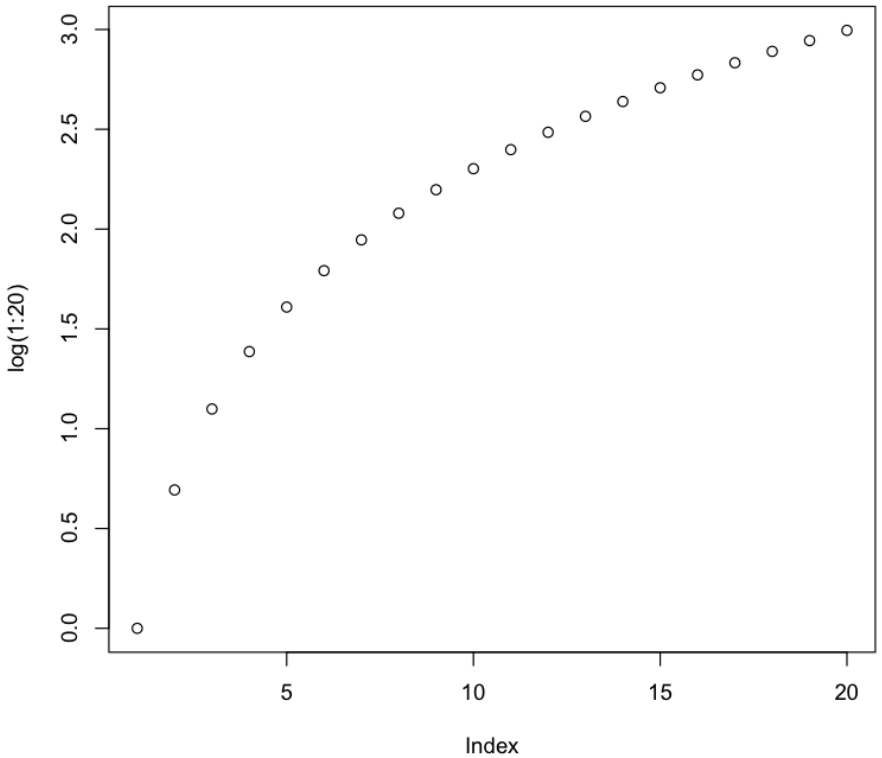
# Introducción a R

¿Qué es R?
Variables
Funciones
Control de flujo
Manejo de gráficas
plots
histogramas

# Gráficas

```
> plot(exp(1:20))
> plot(log(1:20))
```



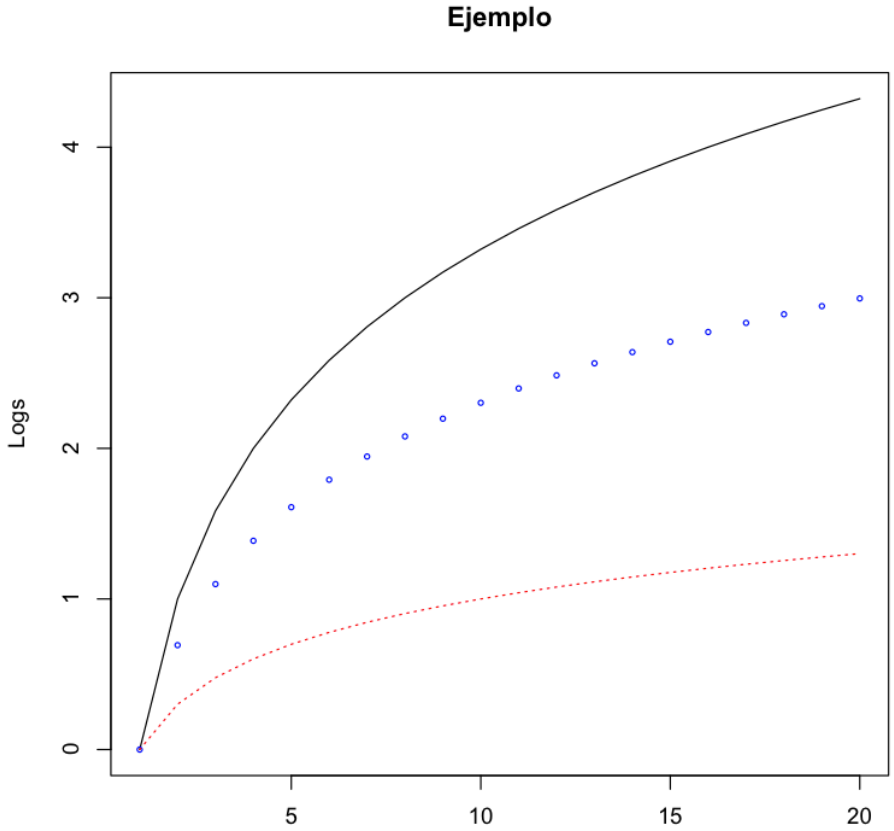



# Gráficas

| Opción | Descripción | Por defecto |
|---|---|---|
| `type` | Tipo de dibujo (“p” para puntos, “l” para líneas, “b” para ambos, etc.) | `“p”` |
| `col` | Color del dibujo (“black”, “red”, “blue”, etc. Consultar colores disponibles con colors()) | `“black”` |
| `xlab` | Título del eje x | `el índice` |
| `ylab` | Título del eje y | `la expresión` |
| `main` | Título de la gráfica | `ninguno` |
| `cex` | Tamaño de los puntos | `1` |
| `lty` | Tipo de líneas (1 continua, 2 discontinua…) | `1` |
| **`?par`** | para consultar éstas y otras muchas opciones | **`-`** |

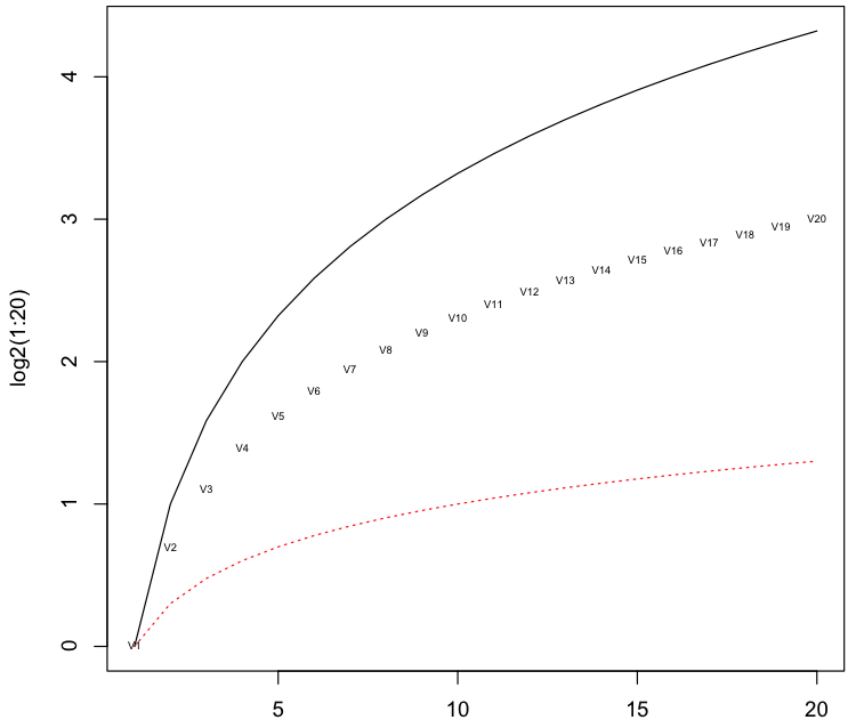
- Sobre una gráfica podemos superponer otras (`points` y `lines`)

```
> plot(log2(1:20), type="l", main="Ejemplo", ylab="Logs")
> points(log(1:20), col="blue", cex=0.5)
> lines(log10(1:20), col="red", lty=3)
```



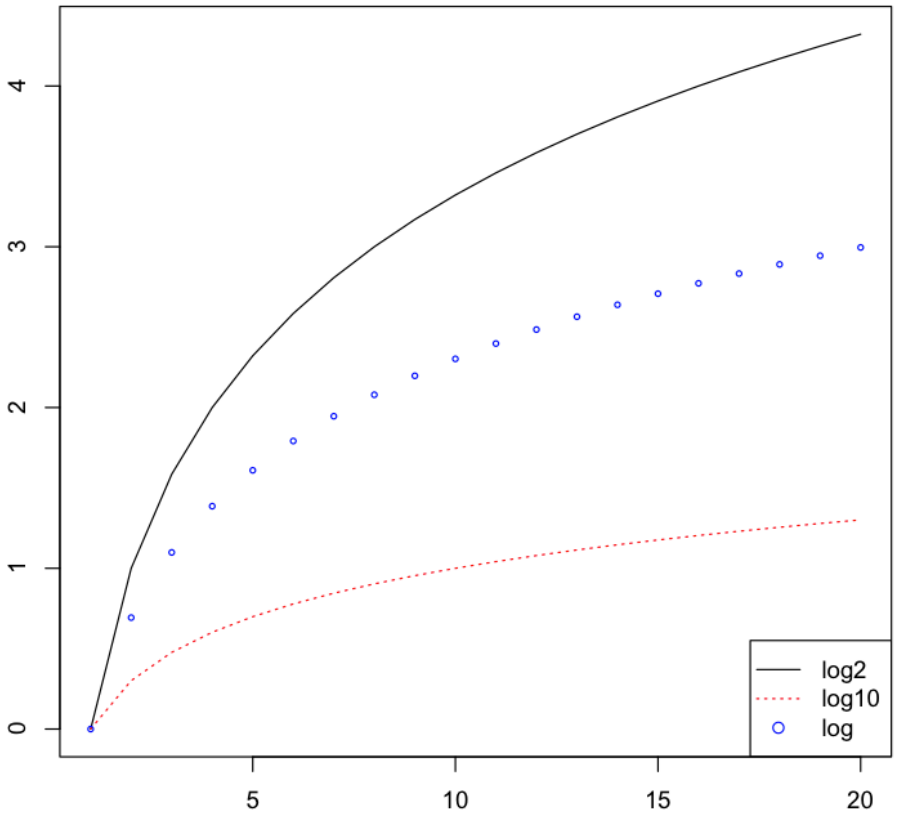
# Graficas (texto)

```
> plot(log2(1:20), type="l")
> lines(log10(1:20), col="red", lty=3)
> text(x=log(1:20), labels=paste("V", c(1:20), sep=""),cex=0.5)
```



# Gráficas (leyenda)

```
legend(x="bottomright”,
       legend=c("log2", "log10", "log"),
       lty=c(1,3,-1),
       pch=c(-1, -1, 1),
       col=c("black", "red", "blue"))
```




- Parámetros
  - x – posición
    - Acepta números o texto tipo “bottomright”, “topleft”, etc.
  - legend – texto
  - col – color
  - pch – tipo de punto
    - -1 si es una línea
  - lty – tipo de línea
    - -1 si son puntos

# Histogramas

- Diagramas de frecuencia de valores
  - Indica cuántos valores "caen" en cada intervalo mediante barras
- Mismos argumentos que `plot`