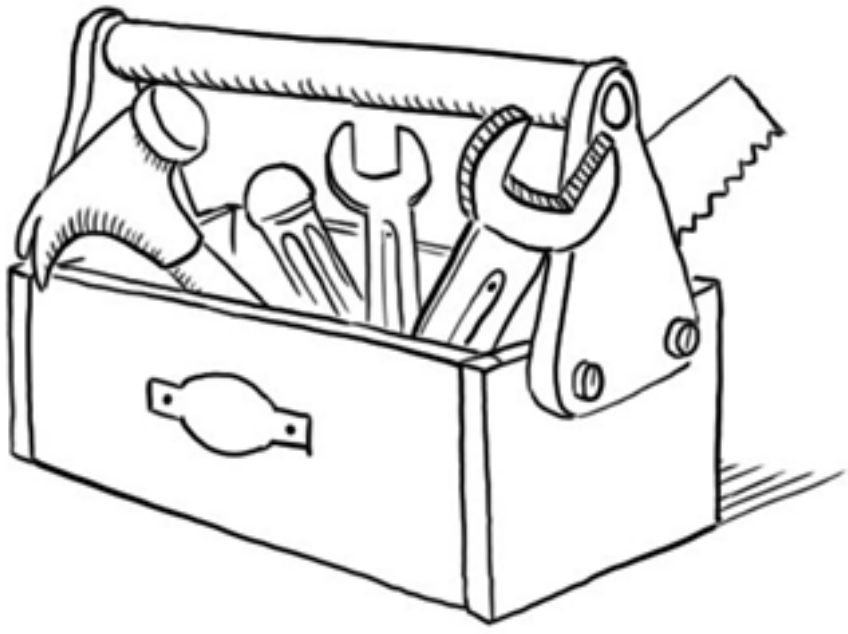
Caja de Herramientas: R@FSOC

TAO

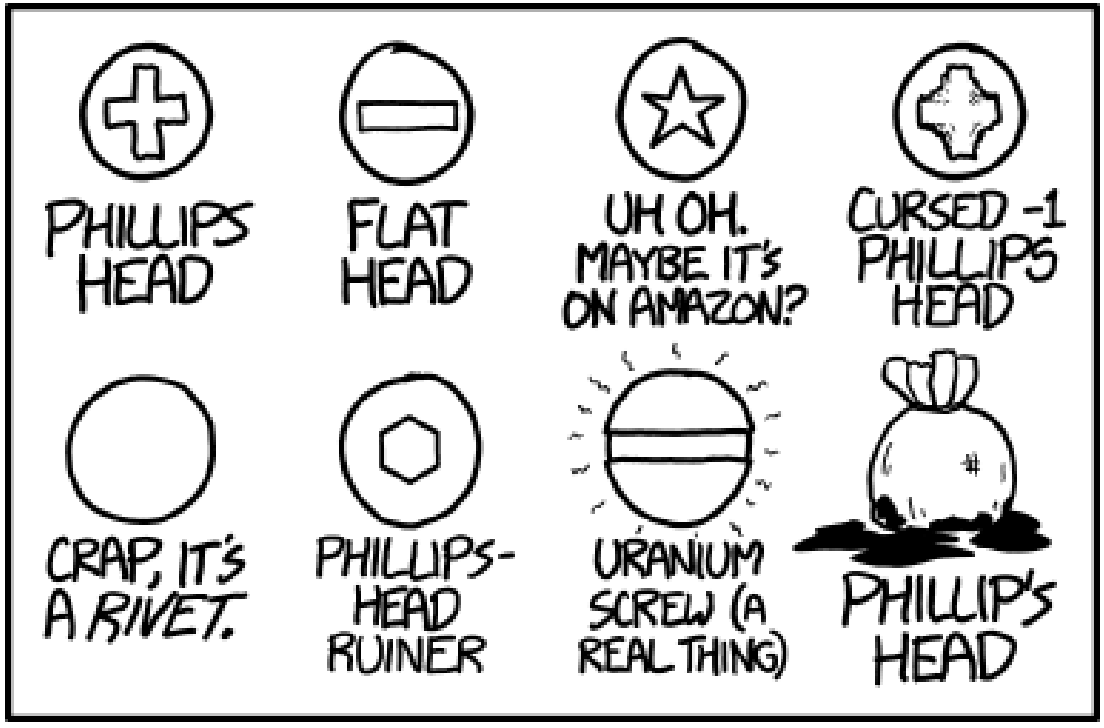
5/14/2021

Caja de Herramientas



Presentación del Problema

Mundo Tabla



¿Es el Mundo Tabla la herramienta para todos nuestros problemas?

Mundo Tabla

Mundo Tabla: ¿Cual es el sentido de la vida, el universo y todo lo demás?

42

Mundo Tabla

Sabiendo que, un número primo es un número natural mayor que 1 que tiene únicamente dos divisores positivos distintos: él mismo y el 1.

Mundo Tabla: ¿Cual es en número primo 1000?

...

Rant

Sintaxis de R

Expresiones

El código R está compuesto por una serie de *expresiones*.

Ejemplos:

```
# Instrucción condicional
if (1 > 2) "mayor" else "menor"
```

```
## [1] "menor"
```

Ejemplos:

```
# Instrucción asignación
x <- 1
```

Instrucciones de asignación

Creamos objetos con el operador de asignación. Las instrucciones de asignación tienen la forma:

- `nombre_objeto <- valor`

```
# Instrucción asignación
x <- 1
```

Expresiones

R proporciona diferentes construcciones para agrupar expresiones:

- `punto y coma`
- `paréntesis`
- `llaves`

Tipos básicos en R

Tipos básicos en R

R tiene cuatro tipos básicos:

- logical
- numeric
- integer
- character

R proporciona cuatro tipos básicos de datos, también conocidos como vectores atómicos.

Logical

El tipo logical es la forma que tiene R para los datos binarios. Usados en test logicos son conocidos como valores boleanos y toman los valores TRUE y FALSE. TRUE y FALSE pueden ser abreviados con las T y F en mayúsculas respectivamente, como podemos ver en el ejemplo, Sin embargo, te recomiendo que utilices la versión completa de TRUE y FALSE.

- Valores booleanos, pueden ser TRUE o FALSE
- Usados en condiciones lógicas

```
3 < 4
```

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

```
class(TRUE)
```

```
## [1] "logical"
```

```
class(T)
```

```
## [1] "logical"
```

Logical

Es posible construir condiciones lógicas utilizando los operadores lógicos y de comparación.

```
a <- 2
b <- 4
a == b # ¿es igual a b?
```

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

```
a != b # ¿es a distindo de b?
```

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

```
(a <3) & (b < 5)  # ¿es a menor que 3 y b menor que 3?
```

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

```
(a < 1) | (b < 3)  # ¿es a menor que 1 o b menor que 3?
```

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

Operadores de Comparación

	?Comparision
$<$	Menor que
$>$	Mayor que
$==$	Igual a
$<=$	Menor que o igual a
$>=$	Mayor que o igual a
$!=$	Distinto a
%in%	Pertenece a
is.na()	Es el valor NA?
!is.na()	Valores distintos a NA

Operadores Lógicos

	?base::Logic
&	y *boleano*
|	o *boleano*
!	no
any	Cualquiera verdadero
all	Todos verdaderos

Para mas información sobre la sintaxis de los operadores y su precedencia consultar la documentación R:

```
# Sintaxis de comparación
?Comparison
# Operadores lógicos
?base::Logic
```

Numeric

- Usados para números
- Apropiados para matemáticas

Para representar los numeros reales R proporciona el tipo *numeric*. Podemos realizar toda clase de operaciones con ellos como por ejemplo sumas, restas, multiplicaciones, divisiones y todo tipo de calculos.

```
mi_altura_en_cm <- 174
mi_altura_en_cm
```

```
## [1] 174
```

```
mi_edad <- 33
mi_edad
```

```
## [1] 33
```

Integer

- Un tipo especial de **numeric** es el **integer**
- Añadir la letra "L"

Un tipo especial de numeric es el integer. Este es el modo que tiene R de representar los numeros enteros. Para especificar que un numero es natural, debemos añadir la letra L en mayúscula como sufijo.

```
mi_edad <- 40L
mi_edad
```

```
## [1] 40
```

En el siguiente ejemplo, podemos apreciar la diferencia entre el numero real y el numero natural por medio de la función class.

```
class(40)
```

```
## [1] "numeric"
```

```
class(40L)
```

```
## [1] "integer"
```

Character

Cualquier carácter/cadena de carácteres entre comillas sera interpretado por R como "character"

```
"Ciencia Política"
```

```
## [1] "Ciencia Política"
```

```
x <- "Ciencia Política"
x
```

```
## [1] "Ciencia Política"
```

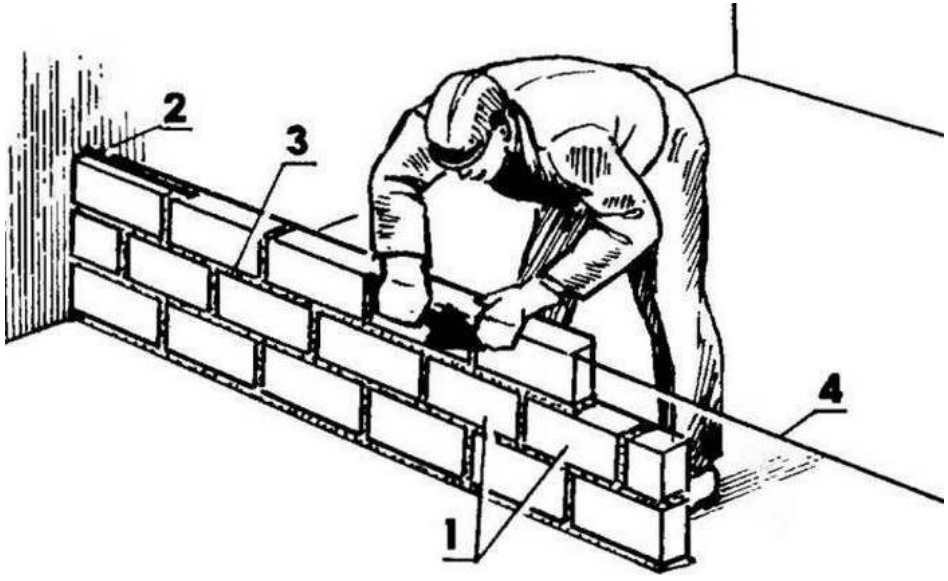
Otros tipos de datos básicos

Por último, cabe mencionar que existen otros tipos de datos básicos en R, como el double que es un tipo numerico con mayor precisión que el numeric. El complex para números complejos y el raw para almacenar bytes.Sin embargo, no los trataremos en este curso puesto que raramente se utilizan en el análisis de datos

- Double (más precisión que numeric)
- Complex (números complejos)
- Raw (almacenar bytes)

Objetos de R

Objetos de R ($\sim\sim$ Legos)



Objetos de R ($\sim\sim$ Legos)

En R, las variables no necesitan ser declaradas como un tipo de datos. Las variables son asignadas a objetos de R y el tipo de datos del objeto de R se sonvierte en el tipo de datos de la Variable. Cabe destacar los siguientes objetos de R:

Vectores

```r
# Crear a vector.
manzana <- c('roja','verde',"amarilla")
print(manzana)
```

```
## [1] "roja"     "verde"    "amarilla"
```

```r
# Class del vector.
print(class(manzana))
```

```
## [1] "character"
```

Listas

```r
# Crear una list.
list1 <- list(c(2,5,3),21.3,sin)
print(list1)
```

```
## [[1]]
## [1] 2 5 3
##
## [[2]]
## [1] 21.3
##
## [[3]]
## function (x)  .Primitive("sin")
```

Matrices

```r
# Crear una matrix.
M = matrix( c('a','a','b','c','b','a'),
            nrow = 2, ncol = 3, byrow = TRUE)
print(M)
```

```
##      [,1] [,2] [,3]
## [1,] "a"  "a"  "b"
## [2,] "c"  "b"  "a"
```

Arrays

```r
# Crear an array.
a <- array(c('verde','amarillo'),dim = c(3,3,2))
print(a)
```

```
## , , 1
##
##      [,1]       [,2]       [,3]
## [1,] "verde"    "amarillo" "verde"
## [2,] "amarillo" "verde"    "amarillo"
## [3,] "verde"    "amarillo" "verde"
##
## , , 2
##
##      [,1]       [,2]       [,3]
## [1,] "amarillo" "verde"    "amarillo"
## [2,] "verde"    "amarillo" "verde"
## [3,] "amarillo" "verde"    "amarillo"
```

Factors

```r
# Create a vector.
manzanas <- c('verde','roja','amarilla',
              'roja','amarilla','verde')

# Crear un  objeto factor.
factor_manzanas <- factor(manzanas)

# Caracterización del factor.
str(factor_manzanas)
```

```
##  Factor w/ 3 levels "amarilla","roja",..: 3 2 1 2 1 3
```

Data Frames

```r
# Create the data frame.
BMI <-  data.frame(
   gender = c("H", "H","F"),
   height = c(152, 171.5, 165),
   weight = c(81,93, 78),
   Age = c(42,38,26)
)
print(BMI)
```

```
##   gender height weight Age
## 1      H  152.0     81  42
## 2      H  171.5     93  38
## 3      F  165.0     78  26
```

Estructuras de Datos

Las colecciones o conjunto de datos en R se organizan por su dimensión (1º, 2º, o varias dimensiones) y si son homogéneas (todos los objetos deben ser del mismo tipo) o heterogéneas (el contenido puede ser de diferentes tipos). A continuación mostramos los cinco tipos de datos más usados en el análisis de datos:

	Homogénea	**Heterogénea**
1	Vector atómico	Lista
2	Matriz	Data frame
n	Array	

Tabla 1 Estructuras de datos

Estructuras de Datos

Como veremos podemos seleccionar un único elemento o varios elementos, mediante el uso de la notación de índices que proporciona R. Se puede elegir elementos por localización dentro de una estructura o por nombre.

La Tabla 2 resume los operadores que aporta R para el acceso a objetos en estructuras de datos.

Sintaxis	**Objetos**	**Descripción**
`x[i]`	Vectores, Listas	Selecciona elementos del objeto *x*, descritos en *i*. *i* puede ser un vector de tipo integer, chararacter (de nombres de los objetos) o lógico. Cuando es usado con listas, devuelve una lista. Cuando es usado en vectores devuelve un vector.
`x[[i]]`	Listas	Devuelve un único elemento de *x* que se encuentra en la posición *i*. *i* puede ser un vector de tipo integer o character de longitud 1.
`x$n`	Listas, Dataframes	Devuelve un objeto con nombre *n* del objeto *x*.
`[i, j]`	Matrices	Devuelve el objeto de la fila *i* y columna *j*. *i* y *j* pueden ser un vector de tipo integer o chararacter (de nombres de los objetos)

Tabla 2 Notación para acceder estructuras de datos

Mundo Tidy

Instalar y Cargar un Paquete

```
install.packages("tidyverse")
```

```
## Installing package into '/home/tao/R/x86_64-pc-linux-gnu
## (as 'lib' is unspecified)
```

```
library(tidyverse)
```

```
## -- Attaching packages ---------------------------------
## v ggplot2 3.3.3     v purrr   0.3.4
## v tibble  3.1.0     v dplyr   1.0.5
## v tidyr   1.1.3     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.1
## -- Conflicts ------------------------------------------
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

Tibble

```
dat <- tibble(
    u = letters[1:5], v = as.factor(letters[1:5]),
    w = 1:5,          x = runif(5,-1,1),
    y = letters[1:5], z = x>0,
    omega = 1+3i
)
```

u	v	w	x	y	z	omega
a	a	1	-0.3787559	a	FALSE	1+3i
b	b	2	0.1886361	b	TRUE	1+3i
c	c	3	0.1006154	c	TRUE	1+3i
d	d	4	-0.6355016	d	FALSE	1+3i
e	e	5	0.8364423	e	TRUE	1+3i

¿Qué es un tibble?

```
## tibble [5 x 7] (S3: tbl_df/tbl/data.frame)
##  $ u    : chr [1:5] "a" "b" "c" "d" ...
##  $ v    : Factor w/ 5 levels "a","b","c","d",..: 1 2 3 4 5
##  $ w    : int [1:5] 1 2 3 4 5
##  $ x    : num [1:5] -0.379 0.189 0.101 -0.636 0.836
##  $ y    : chr [1:5] "a" "b" "c" "d" ...
##  $ z    : logi [1:5] FALSE TRUE TRUE FALSE TRUE
##  $ omega: cplx [1:5] 1+3i 1+3i 1+3i ...
```

Tipo de Datos

- *dbl* significa dobles, o números reales.
- *chr* significa vectores de caracteres o cadenas.
- *dttm* significa fechas y horas (una fecha + una hora).
- *lgl* significa lógico, vectores que solo contienen TRUE (verdadero) o FALSE (falso).
- *fctr* significa factores, que R usa para representar variables categóricas con valores posibles fijos.
- *date* significa fechas.
- *int* significa números enteros

Extraer Columnas

```
df <- tibble(
x = runif(5),
y = rnorm(5)
)

df
```

```
## # A tibble: 5 x 2
##       x      y
##   <dbl>  <dbl>
## 1 0.882  1.28
## 2 0.188 -0.330
## 3 0.723 -1.03
## 4 0.381 -0.602
## 5 0.911  1.75
```

Extraer Columnas a Vector

Por nombre

```
df[["x"]]
```

```
## [1] 0.8824868 0.1877816 0.7230797 0.3810384 0.9109763
```

```
df$x
```

```
## [1] 0.8824868 0.1877816 0.7230797 0.3810384 0.9109763
```

Por posicion

```
df[[1]]
```

```
## [1] 0.8824868 0.1877816 0.7230797 0.3810384 0.9109763
```

Usando pipes

```
df %>% .$x
```

```
## [1] 0.8824868 0.1877816 0.7230797 0.3810384 0.9109763
```

```
df %>% .[["x"]]
```

```
## [1] 0.8824868 0.1877816 0.7230797 0.3810384 0.9109763
```

Extraer Columnas manteniendo el D.F.

Por nombre

```
df["x"]
```

```
## # A tibble: 5 x 1
##       x
##   <dbl>
## 1 0.882
## 2 0.188
## 3 0.723
## 4 0.381
## 5 0.911
```

Extraer Columnas manteniendo el D.F.

Por posicion

```
df[1]
```

```
## # A tibble: 5 x 1
##       x
##   <dbl>
## 1 0.882
## 2 0.188
## 3 0.723
## 4 0.381
## 5 0.911
```

Extraer Columnas manteniendo el D.F.

Usando pipes

```
df %>% .["x"]
```

```
## # A tibble: 5 x 1
##       x
##   <dbl>
## 1 0.882
## 2 0.188
## 3 0.723
## 4 0.381
## 5 0.911
```

Caja de Herramientas