Parte 3: Slot Machine

## **Capìtulo 7:** Programs

El siguiente còdigo genera tres sìmbolos comunes en este tipo de juegos: diamonds (DD), sevens (7), triple bars (BBB), double bars (BB), single bars (B), cherries (C), and zeroes (0) Los sìmbolos son sleccionados aleatoriamente y cada simbolo aparece con diferentes probabilidades

#Devuelve los valores de la jugada actual  
get\_symbols <- function() {  
 wheel <- c("DD", "7", "BBB", "BB", "B", "C", "0")  
 sample(wheel, size = 3, replace = TRUE,  
 prob = c(0.03, 0.03, 0.06, 0.1, 0.25, 0.01, 0.52))  
}

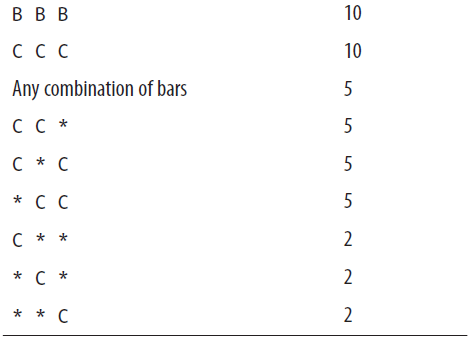
get\_symbols() #Devuelve 3 valores diferentes en cada ejecuciòn

## [1] "0" "0" "0"

Un jugador gana un premio si obtiene:

1. Tres ìmbolos iguales
2. Tres barras (De cualquier tipo
3. Una o màs cerezas





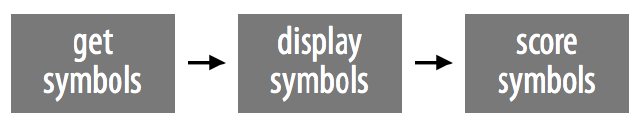
Los programas en R pueden ser guardads como Scripts o como funciones.

Se crearà la funciòn “score” para calcular el premio

#Funcion que simula la slot machine  
play <- function() {  
 symbols <- get\_symbols()  
 print(symbols)  
 score(symbols)  
}

Lo ideal es dividir los còdigos en pequeñas tareas para facilitar la programaciòn. hay dos tipos de estas subtareas en R. sequential steps y parallel cases

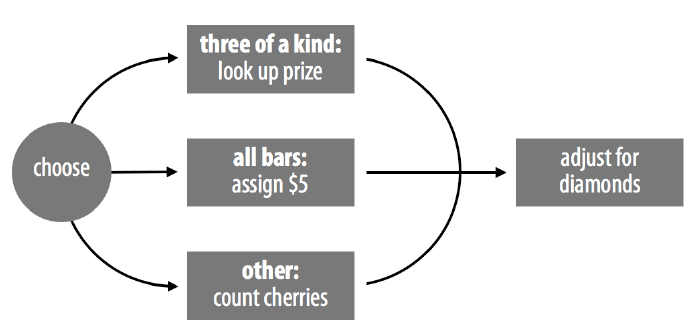
### Sequential steps



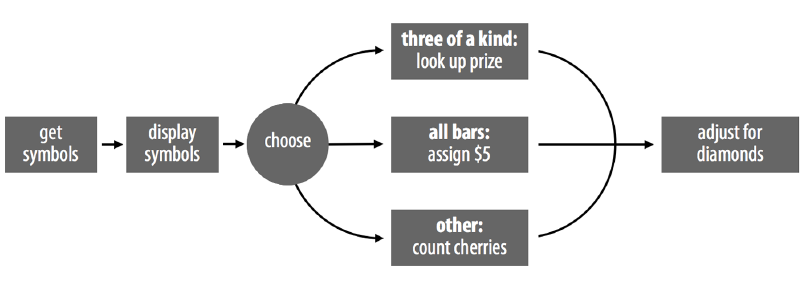
un ejemplo de esto es la funciòn play() que se programò anteriormente

play <- function() {  
 # step 1: Genera los sìmbolos  
 symbols <- get\_symbols()  
 # step 2: Muestra los sìmbolos  
 print(symbols)  
 # step 3: Puntea los sìmbolos  
 score(symbols)  
}

### Parallel Cases



El procesopuede incluir ambos tipos de subtareas



### if Statements

Las subtareas en paralelos se crearàn con la ayuda de los condiocionales. La lògica de los condicionales en R es parecido a lo siguiente:

if (this) {

that

}

#Si num es negativo entonces el valor es multiplicado por -1  
num=-5  
if (num < 0) {  
 num <- num \* -1  
}

num <- -1  
if (num < 0) {  
 print("num is negative.")  
 print("Don't worry, I'll fix it.")  
 num <- num \* -1  
 print("Now num is positive.")  
}

## [1] "num is negative."  
## [1] "Don't worry, I'll fix it."  
## [1] "Now num is positive."

num

## [1] 1

x <- 1  
 if (3 == 3) {  
 x <- 2  
}  
x

## [1] 2

x <- 1  
 if (TRUE) { #La condiciòn ya de por sì es verdadera asì que no necesita evaluarla y automàticamente se entrarà al còdigo del condicional  
 x <- 2  
}  
x

## [1] 2

x <- 1  
if (x == 1) {  
 x <- 2  
 if (x == 1) {  
 x <- 3  
 }  
}  
x

## [1] 2

### Else Statements

Lo que debe el còdigo si el resultado es falso

if (this) {

Plan A

} else {

Plan B

}

a <- 0.14  
dec <- a - trunc(a) #obtengo los valores decimales de un nùmero  
dec

## [1] 0.14

#Si el nùmero es mayor a 0,5   
if (dec >= 0.5) {  
 a <- trunc(a) + 1  
} else {  
 a <- trunc(a)  
}  
a

## [1] 0

a <- 12  
b <- 13  
  
if (a > b) {  
 print("A wins!")  
} else if (a < b) {  
 print("B wins!")  
} else {  
 print("Tie.")  
}

## [1] "B wins!"

La siguiente es la estructrura que seguirà el còdigo para la puntuaciòn de la màquina:[

if ( # Case 1: all the same ) { (((1)))

prize <- # look up the prize``(((2)))

} else if ( # Case 2: all bars ) {``(((3)))

prize <- # assign $5``(((4)))

} else {

# count cherries prize``(((5)))

<- # calculate a prize``(((6)))

}

# count diamonds``(((7)))

# double the prize if necessary``(((8)))

(((1))) Prueba si los tres simbolos son de mismo tipo

(((3))) Prueba si los simbolos son todos barras

(((2))) Busca el premio por tres del mismo tipo basado en el símbolo común

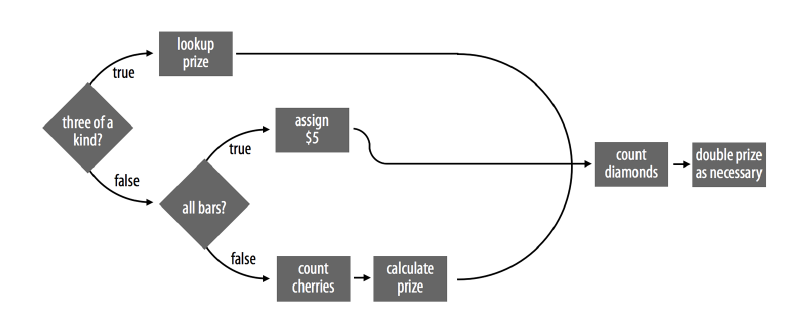
(((4))) Asigna un premio de 5

(((5))) Cuenta el nùmero de cerezas

(((6))) Cuenta el nùmero de diamantes

(((7))) Calcula un premio basado en el número de cerezas.

(((8))) Ajusta el premio para los diamantes



Cada uno de los bloquees corresponde a una subtarea dentro del programa. Se puede tratar de programar subtarea por subtarea.

la funciòn final lucira parecido a la siguiente

score <- function(symbols){  
 #Calcula un precio  
 prize  
}

Para cada sutarea lo mejor es crear un ejemplo con el fin de probarla y no tener que programar toda la funciòn de golpe. Lo que facilita que todo el còdigo funcione correctamente por completo.

symbols <- c("7", "7", "7") # Con este codigo ya puedo correr algunas subtareas

Como en algunas subtareas no funcionarà podemos usar los siguientes

symbols <- c("B", "BB", "BBB")

symbols <- c("C", "DD", "0")

#### Primer caso

Prueba si los tres simbolos son de mismo tipo

symbols

## [1] "C" "DD" "0"

symbols[1] == symbols[2] & symbols[2] == symbols[3]

## [1] FALSE

symbols[1] == symbols[2] & symbols[1] == symbols[3]

## [1] FALSE

all(symbols == symbols[1])

## [1] FALSE

length(unique(symbols) == 1)

## [1] 3

#unique devuelve los valores ùnicos del vector (Elimina los duplicados)  
#length calcula el nùmero de elementos dentro del vector  
#En este caso esta instruccion devuelve el valor de 1 si todos los datos del vector son iguales  
  
  
#Crearemos la variable "same" paraa la primer subtarea  
same=same <- symbols[1] == symbols[2] && symbols[2] == symbols[3]

score <- function(symbols){  
 if(symbols[1] == symbols[2] & symbols[2] == symbols[3]){  
 prize <- "Iguales"  
 }  
 else{  
 prize <- "Diferentes"  
 }  
}  
score(symbols)

#### Segundo caso

El vector tiene soo simbolos del tipo barra

symbols <- c("B", "BB", "BBB")

symbols[1] == "B" | symbols[1] == "BB" | symbols[1] == "BBB" & symbols[2] == "B" | symbols[2] == "BB" | symbols[2] == "BBB" & symbols[3] == "B" | symbols[3] == "BB" | symbols[3] == "BBB"

## [1] TRUE

all(symbols %in% c("B", "BB", "BBB")) #Comprueba si todos los valores del vector simbolo son al menos no de los 3 valores del vector

## [1] TRUE

#Crearemos la variable "bars" paraa la segunda subtarea  
same=same <- symbols[1] == symbols[2] && symbols[2] == symbols[3]

Se puede observar que algunos datos del caso uno pueden cumplir tambièn con el caso 2

symbols <- c("B", "B", "B")  
all(symbols %in% c("B", "BB", "BBB"))

## [1] TRUE

#Pero el condicional saltarà el caso 2 si el caso 1 se cumple asì que nunca se evaluara por defecto ese valor

#### Tercer caso

Esta subtarea establece el valor de la variable prize, lo cual depende de los sìmbolos que se contengan. Lo que generarà el siguiente còdigo

if (same) {  
 symbol <- symbols[1]  
 if (symbol == "DD") {  
 prize <- 100  
 } else if (symbol == "7") {  
 prize <- 80  
 } else if (symbol == "BBB") {  
 prize <- 40  
 } else if (symbol == "BB") {  
 prize <- 5  
 } else if (symbol == "B") {  
 prize <- 10  
 } else if (symbol == "C") {  
 prize <- 10  
 } else if (symbol == "0") {  
 prize <- 0  
 }  
}

### Lookup Tables

Es usuar trabajar subconjuntos en R. Para eso puedo crear un vector que capture a relaciòn entre los sìmbolos y sus premios. Este vector puede almacenar símbolos como nombres y valores de premios como elementos.

payouts <- c("DD" = 100, "7" = 80, "BBB" = 40, "BB" = 25, "B" = 10, "C" = 10, "0" = 0)  
payouts

## DD 7 BBB BB B C 0   
## 100 80 40 25 10 10 0

payouts["DD"] #Me devuelven los valores correspondientes que posee el vector

## DD   
## 100

payouts["B"]

## B   
## 10

unname(payouts["DD"]) #Me oculta el nombre del dato y me muestra solo el valor

## [1] 100

#Esto funciona para traer el valor del dato que necesito saber  
symbols <- c("7", "7", "7")  
symbols[1]

## [1] "7"

payouts[symbols[1]]

## 7   
## 80

symbols <- c("C", "C", "C")  
payouts[symbols[1]]

## C   
## 10

#Este paso me ahorra el condicional anterior  
  
#Entonces para el tercer caso tenemos de salida:  
payouts <- c("DD" = 100, "7" = 80, "BBB" = 40, "BB" = 25,"B" = 10, "C" = 10, "0" = 0)  
prize <- unname(payouts[symbols[1]])

#### Ultimo caso

Determinar el nùmero de cherriesque hay en los sìmbolos

symbols <- c("C", "DD", "C")  
symbols == "C"

## [1] TRUE FALSE TRUE

#A la salida de este caso obtenemos  
cherries = sum(symbols == "C") #En este caso por ls reglas de coercion, R cinvertirà los valores booleanos en 1 y 0 y al realizar la suma me devolverà el nùmero de datos que son verdaderos  
sum(symbols == "DD")

## [1] 1

#Teniendo la cantidad de cherries podemos ahora sì aplicar la condiciòn para cada cantidad  
  
if (cherries == 2) {  
 prize <- 5  
} else if (cherries == 1) {  
 prize <- 2  
} else {  
 prize <- 0  
}  
  
#Pero este còdigo es algo extenso y puede acortarse màs

Para este caso la mejor soluciòn incluirà el subsetting

c(0, 2, 5) #Se crea un vector que contenga la informaciòn de los precios

## [1] 0 2 5

c(0, 2, 5)[1] #Elige el primer valor del vector

## [1] 0

#Como el valor cherries puede llegar a valer cero, y dentro de lso vectores no existe la posiciòn cero, sumamos el +1 a cada valor de cherries.Pasarìa de contener 0,1,2 a contener 1,2,3 aue son las posiciones de los datos  
  
cherries + 1

## [1] 3

c(0, 2, 5)[cherries + 1] #Devuelve el valor de la tercera posiciòn

## [1] 5

Unificando el còdigo completo tenemos:

same <- symbols[1] == symbols[2] && symbols[2] == symbols[3]  
bars <- symbols %in% c("B", "BB", "BBB")  
  
if (same) {  
 payouts <- c("DD" = 100, "7" = 80, "BBB" = 40, "BB" = 25, "B" = 10, "C" = 10, "0" = 0)  
 prize <- unname(payouts[symbols[1]])  
} else if (all(bars)) {  
 prize <- 5  
} else {  
 cherries <- sum(symbols == "C")  
 prize <- c(0, 2, 5)[cherries + 1]  
}  
  
 diamonds <- sum(symbols == "DD")

La ùltima subtarea solo consiste en duplicar el precio por cada diamente presente.

SI no hay ningùn diamante presente:  
prize \* 1 # 1 = 2 ^ 0

Si hay uno  
prize \* 2 # 2 = 2 ^ 1

dos  
prize \* 4 # 4 = 2 ^ 2

Tres  
prize \* 8 # 8 = 2 ^ 3

Codigo final:

score <- function(symbols) {  
 # identify case  
 same <- symbols[1] == symbols[2] && symbols[2] == symbols[3]  
 bars <- symbols %in% c("B", "BB", "BBB")  
 # get prize  
 if (same) {  
 payouts <- c("DD" = 100, "7" = 80, "BBB" = 40, "BB" = 25, "B" = 10, "C" = 10, "0" = 0)  
 prize <- unname(payouts[symbols[1]])  
 } else if (all(bars)) {  
 prize <- 5  
 } else {  
 cherries <- sum(symbols == "C")  
 prize <- c(0, 2, 5)[cherries + 1]  
 }  
 # adjust for diamonds  
 diamonds <- sum(symbols == "DD")  
 prize \* 2 ^ diamonds  
}

Y la funciòn play por fìn podrà funcionar

play <- function() {  
symbols <- get\_symbols()  
print(symbols)  
score(symbols)  
}

play()

## [1] "B" "0" "0"

## [1] 0

## CHAPTER 8: S3

La slot machine queriamos que arrojara resultados en el sguiente formato

play()  
## 0 0 DD  
## $0

Pero en cambio estamos teniendo lo siguiente:

play()  
## “0” “0” “DD”  
## 0

Otro problema es que si intengo guardar los sìmbolos en una variable, el resultado de la variable difiere del resultado entregado por la funciòn

one\_play <- play()

## [1] "BB" "0" "BB"

## "B" "0" "B"  
one\_play

## [1] 0

## 0

### The S3 System

Gobierna como R maneja objetos de diferentes clases.

La funciòn print se comporta de la siguiente forma:

num <- 1000000000 #Cuando tengo un vector numèrico, print mostrarà un nùmero  
print(num)

## [1] 1e+09

## 1000000000  
  
class(num) <- c("POSIXct", "POSIXt") #la clase POSIXct seguida de POSIXt imprimirà una fecha  
print(num)

## [1] "2001-09-08 20:46:40 -05"

## "2001-09-08 19:46:40 CST"  
  
#Lo anterior funcion dentro del S3 system

R’s S3 system is built around three components: attributes (especially the class attribute), generic functions, and methods

#### Attributes

Attributes do not affect the values of the object, but stick to the object as a type of metadata that R can use to handle the object. For example, a data frame stores its row and column names as attributes. Data frames also store their class, “data.frame”, as an attribute.

attributes(one\_play) #Me devuelve los atributos del objeto (tambièn puedo usarla par agregar atributos)

## NULL

attr takes two arguments: an R object and the name of an attribute (as a character string).

attr(one\_play, "symbols") <- c("B", "0", "B") #Establece el atributo symbols   
attributes(one\_play)

## $symbols  
## [1] "B" "0" "B"

para ver solo el atributo en epecífico

attr(one\_play, "symbols")

## [1] "B" "0" "B"

one\_play

## [1] 0  
## attr(,"symbols")  
## [1] "B" "0" "B"

one\_play+1

## [1] 1  
## attr(,"symbols")  
## [1] "B" "0" "B"

A continuación se modifica la función play para devolver un precio que contiene los simbolos asociados con él como un atributo llamado symbols. esto con el fin de eliminar la redundancia con el print(symbols)

play <- function() {  
 symbols <- get\_symbols()  
 print(symbols)  
 score(symbols)  
}

play <- function() {  
 symbols <- get\_symbols()  
 prize <- score(symbols) #Extraigo el precio de los simbolos  
 attr(prize, "symbols") <- symbols #asigno el atributo a la variable precio  
 prize  
}  
play()

## [1] 0  
## attr(,"symbols")  
## [1] "0" "0" "B"

#si asigno el resultado a un nueovo objeto esta vez si se conservará el valor  
two\_play <- play()  
two\_play

## [1] 0  
## attr(,"symbols")  
## [1] "0" "B" "0"

También se pueden establecer los atributos usando la función structure de la siguiente forma

play <- function() {  
 symbols <- get\_symbols()  
 structure(score(symbols), symbols = symbols)  
}  
three\_play <- play()  
three\_play

## [1] 0  
## attr(,"symbols")  
## [1] "0" "0" "7"

Se pueden escribir funciones propias que busquen y utilicen el atributo. Por ejemplo la siguiente fnción buscará y mostrará el atributo de forma más atractiva

slot\_display <- function(prize){  
 # extract symbols  
 symbols <- attr(prize, "symbols")  
 # collapse symbols into single string  
 symbols <- paste(symbols, collapse = " ")  
 # combine symbol with prize as a regular expression  
 # \n is regular expression for new line (i.e. return or enter)  
 string <- paste(symbols, prize, sep = "\n$")  
 # display regular expression in console without quotes  
cat(string)  
}  
slot\_display(one\_play)

## B 0 B  
## $0

La funcón se vería de la siguiente forma paso a paso

symbols <- attr(one\_play, "symbols")  
symbols

## [1] "B" "0" "B"

symbols <- paste(symbols, collapse = " ")  
symbols

## [1] "B 0 B"

prize <- one\_play  
string <- paste(symbols, prize, sep = "\n$")  
string

## [1] "B 0 B\n$0"

cat(string)

## B 0 B  
## $0

slot\_display(play())

## BB 0 0  
## $0

Esta forma de limiar el formato requiere que intervenga de orma manual al construir las funciones. Hay otra forma de hacerlo. La función es **print** y es una función genérica

### Funciones genéricas

print(play())

## [1] 0  
## attr(,"symbols")  
## [1] "0" "B" "B"

num <- 1000000000  
print(num)

## [1] 1e+09

## 1000000000

class(num) <- c("POSIXct", "POSIXt")  
print(num)

## [1] "2001-09-08 20:46:40 -05"

## "2001-09-08 19:46:40 CST"

### Methods

print.POSIXct

## function (x, tz = "", usetz = TRUE, max = NULL, ...)   
## {  
## if (is.null(max))   
## max <- getOption("max.print", 9999L)  
## FORM <- if (missing(tz))   
## function(z) format(z, usetz = usetz)  
## else function(z) format(z, tz = tz, usetz = usetz)  
## if (max < length(x)) {  
## print(FORM(x[seq\_len(max)]), max = max + 1, ...)  
## cat(" [ reached 'max' / getOption(\"max.print\") -- omitted",   
## length(x) - max, "entries ]\n")  
## }  
## else if (length(x))   
## print(FORM(x), max = max, ...)  
## else cat(class(x)[1L], "of length 0\n")  
## invisible(x)  
## }  
## <bytecode: 0x00000225007e1438>  
## <environment: namespace:base>

methods(print)

## [1] print.acf\*   
## [2] print.activeConcordance\*   
## [3] print.AES\*   
## [4] print.anova\*   
## [5] print.aov\*   
## [6] print.aovlist\*   
## [7] print.ar\*   
## [8] print.Arima\*   
## [9] print.arima0\*   
## [10] print.AsIs   
## [11] print.aspell\*   
## [12] print.aspell\_inspect\_context\*   
## [13] print.bibentry\*   
## [14] print.Bibtex\*   
## [15] print.browseVignettes\*   
## [16] print.by   
## [17] print.changedFiles\*   
## [18] print.check\_bogus\_return\*   
## [19] print.check\_code\_usage\_in\_package\*   
## [20] print.check\_compiled\_code\*   
## [21] print.check\_demo\_index\*   
## [22] print.check\_depdef\*   
## [23] print.check\_details\*   
## [24] print.check\_details\_changes\*   
## [25] print.check\_doi\_db\*   
## [26] print.check\_dotInternal\*   
## [27] print.check\_make\_vars\*   
## [28] print.check\_nonAPI\_calls\*   
## [29] print.check\_package\_code\_assign\_to\_globalenv\*   
## [30] print.check\_package\_code\_attach\*   
## [31] print.check\_package\_code\_data\_into\_globalenv\*   
## [32] print.check\_package\_code\_startup\_functions\*   
## [33] print.check\_package\_code\_syntax\*   
## [34] print.check\_package\_code\_unload\_functions\*   
## [35] print.check\_package\_compact\_datasets\*   
## [36] print.check\_package\_CRAN\_incoming\*   
## [37] print.check\_package\_datalist\*   
## [38] print.check\_package\_datasets\*   
## [39] print.check\_package\_depends\*   
## [40] print.check\_package\_description\*   
## [41] print.check\_package\_description\_encoding\*   
## [42] print.check\_package\_license\*   
## [43] print.check\_packages\_in\_dir\*   
## [44] print.check\_packages\_used\*   
## [45] print.check\_po\_files\*   
## [46] print.check\_pragmas\*   
## [47] print.check\_Rd\_line\_widths\*   
## [48] print.check\_Rd\_metadata\*   
## [49] print.check\_Rd\_xrefs\*   
## [50] print.check\_RegSym\_calls\*   
## [51] print.check\_S3\_methods\_needing\_delayed\_registration\*  
## [52] print.check\_so\_symbols\*   
## [53] print.check\_T\_and\_F\*   
## [54] print.check\_url\_db\*   
## [55] print.check\_vignette\_index\*   
## [56] print.checkDocFiles\*   
## [57] print.checkDocStyle\*   
## [58] print.checkFF\*   
## [59] print.checkRd\*   
## [60] print.checkRdContents\*   
## [61] print.checkReplaceFuns\*   
## [62] print.checkS3methods\*   
## [63] print.checkTnF\*   
## [64] print.checkVignettes\*   
## [65] print.citation\*   
## [66] print.cli\_ansi\_html\_style\*   
## [67] print.cli\_ansi\_string\*   
## [68] print.cli\_ansi\_style\*   
## [69] print.cli\_boxx\*   
## [70] print.cli\_diff\_chr\*   
## [71] print.cli\_doc\*   
## [72] print.cli\_progress\_demo\*   
## [73] print.cli\_rule\*   
## [74] print.cli\_sitrep\*   
## [75] print.cli\_spark\*   
## [76] print.cli\_spinner\*   
## [77] print.cli\_tree\*   
## [78] print.codoc\*   
## [79] print.codocClasses\*   
## [80] print.codocData\*   
## [81] print.colorConverter\*   
## [82] print.compactPDF\*   
## [83] print.condition   
## [84] print.connection   
## [85] print.CRAN\_package\_reverse\_dependencies\_and\_views\*   
## [86] print.data.frame   
## [87] print.Date   
## [88] print.default   
## [89] print.dendrogram\*   
## [90] print.density\*   
## [91] print.difftime   
## [92] print.dist\*   
## [93] print.Dlist   
## [94] print.DLLInfo   
## [95] print.DLLInfoList   
## [96] print.DLLRegisteredRoutines   
## [97] print.document\_context\*   
## [98] print.document\_position\*   
## [99] print.document\_range\*   
## [100] print.document\_selection\*   
## [101] print.dummy\_coef\*   
## [102] print.dummy\_coef\_list\*   
## [103] print.ecdf\*   
## [104] print.eigen   
## [105] print.factanal\*   
## [106] print.factor   
## [107] print.family\*   
## [108] print.fileSnapshot\*   
## [109] print.findLineNumResult\*   
## [110] print.formula\*   
## [111] print.ftable\*   
## [112] print.function   
## [113] print.getAnywhere\*   
## [114] print.glm\*   
## [115] print.hashtab\*   
## [116] print.hclust\*   
## [117] print.help\_files\_with\_topic\*   
## [118] print.hexmode   
## [119] print.HoltWinters\*   
## [120] print.hsearch\*   
## [121] print.hsearch\_db\*   
## [122] print.htest\*   
## [123] print.html\*   
## [124] print.html\_dependency\*   
## [125] print.htmltools.selector\*   
## [126] print.htmltools.selector.list\*   
## [127] print.infl\*   
## [128] print.integrate\*   
## [129] print.isoreg\*   
## [130] print.key\_missing\*   
## [131] print.kmeans\*   
## [132] print.knitr\_kable\*   
## [133] print.Latex\*   
## [134] print.LaTeX\*   
## [135] print.libraryIQR   
## [136] print.listof   
## [137] print.lm\*   
## [138] print.loadings\*   
## [139] print.loess\*   
## [140] print.logLik\*   
## [141] print.ls\_str\*   
## [142] print.medpolish\*   
## [143] print.MethodsFunction\*   
## [144] print.mtable\*   
## [145] print.NativeRoutineList   
## [146] print.news\_db\*   
## [147] print.nls\*   
## [148] print.noquote   
## [149] print.numeric\_version   
## [150] print.object\_size\*   
## [151] print.octmode   
## [152] print.packageDescription\*   
## [153] print.packageInfo   
## [154] print.packageIQR\*   
## [155] print.packageStatus\*   
## [156] print.pairwise.htest\*   
## [157] print.person\*   
## [158] print.POSIXct   
## [159] print.POSIXlt   
## [160] print.power.htest\*   
## [161] print.ppr\*   
## [162] print.prcomp\*   
## [163] print.princomp\*   
## [164] print.proc\_time   
## [165] print.quosure\*   
## [166] print.quosures\*   
## [167] print.raster\*   
## [168] print.Rconcordance\*   
## [169] print.Rd\*   
## [170] print.recordedplot\*   
## [171] print.restart   
## [172] print.RGBcolorConverter\*   
## [173] print.RGlyphFont\*   
## [174] print.rlang:::list\_of\_conditions\*   
## [175] print.rlang\_box\_done\*   
## [176] print.rlang\_box\_splice\*   
## [177] print.rlang\_data\_pronoun\*   
## [178] print.rlang\_dict\*   
## [179] print.rlang\_dyn\_array\*   
## [180] print.rlang\_envs\*   
## [181] print.rlang\_error\*   
## [182] print.rlang\_fake\_data\_pronoun\*   
## [183] print.rlang\_lambda\_function\*   
## [184] print.rlang\_message\*   
## [185] print.rlang\_trace\*   
## [186] print.rlang\_warning\*   
## [187] print.rlang\_zap\*   
## [188] print.rle   
## [189] print.rlib\_bytes\*   
## [190] print.rlib\_error\_3\_0\*   
## [191] print.rlib\_trace\_3\_0\*   
## [192] print.roman\*   
## [193] print.SavedPlots\*   
## [194] print.sessionInfo\*   
## [195] print.shiny.tag\*   
## [196] print.shiny.tag.env\*   
## [197] print.shiny.tag.list\*   
## [198] print.shiny.tag.query\*   
## [199] print.simple.list   
## [200] print.smooth.spline\*   
## [201] print.socket\*   
## [202] print.srcfile   
## [203] print.srcref   
## [204] print.stepfun\*   
## [205] print.stl\*   
## [206] print.StructTS\*   
## [207] print.subdir\_tests\*   
## [208] print.summarize\_CRAN\_check\_status\*   
## [209] print.summary.aov\*   
## [210] print.summary.aovlist\*   
## [211] print.summary.ecdf\*   
## [212] print.summary.glm\*   
## [213] print.summary.lm\*   
## [214] print.summary.loess\*   
## [215] print.summary.manova\*   
## [216] print.summary.nls\*   
## [217] print.summary.packageStatus\*   
## [218] print.summary.ppr\*   
## [219] print.summary.prcomp\*   
## [220] print.summary.princomp\*   
## [221] print.summary.table   
## [222] print.summary.warnings   
## [223] print.summaryDefault   
## [224] print.table   
## [225] print.tables\_aov\*   
## [226] print.terms\*   
## [227] print.ts\*   
## [228] print.tskernel\*   
## [229] print.TukeyHSD\*   
## [230] print.tukeyline\*   
## [231] print.tukeysmooth\*   
## [232] print.undoc\*   
## [233] print.vignette\*   
## [234] print.warnings   
## [235] print.xfun\_raw\_string\*   
## [236] print.xfun\_record\_results\*   
## [237] print.xfun\_rename\_seq\*   
## [238] print.xfun\_strict\_list\*   
## [239] print.xgettext\*   
## [240] print.xngettext\*   
## [241] print.xtabs\*   
## see '?methods' for accessing help and source code

Esos métodos en el caso de la función print permiten a dicha funcion comportarse dependiendo de la clase del objeto con la cual esté trabajando.

### Method dispatch

class(one\_play) <- "slots" #Asigna la clase "slots" a la funcion one\_play

El metodo para esta función debe llamarse print.slots

args(print)

## function (x, ...)   
## NULL

print.slots <- function(x, ...){  
 cat("Im using the print.slots method")  
}  
print(one\_play)

## Im using the print.slots method

one\_play

## Im using the print.slots method

rm(print.slots)

Algunos objetos poseen dos clases

now <- Sys.time()  
attributes(now)

## $class  
## [1] "POSIXct" "POSIXt"

UseMethod will first look for a method that matches the first class listed in the object’s class vector. If UseMethod cannot find one, it will then look for the method that matches the second class

Ahora creemos un método para imprimir de mejor forma la slot machine

print.slots <- function(x, ...) {  
 slot\_display(x)  
}  
one\_play #La anterior función provoca que todos los objetos que tengan por clase slots, se mustren automaticamente al ejecutar la instrucción. En este caso no encesito usar print para imprimir one\_play

## B 0 B  
## $0

A cntinuación asignaremos la clase slots a la función play(). Como recuerdo. Lo que hicimos fue crear o estandarizar una forma de impresión. Queremos que print imprima en el formato que queramos. Dicho formato lo establecimos en slot\_display, la cual asignamos como forma de impresión predeterminada a la clase slots, entonces, todos los objetos cuya clase sea slot, se imprimira bajo es formato que se estableció anteriormente.

play <- function() {  
 symbols <- get\_symbols()  
 structure(score(symbols), symbols = symbols, class = "slots")  
}  
  
class(play())

## [1] "slots"

play()

## 0 0 B  
## $0

CODIGO DE RESUMEN

#SE CREÓ LA FUNCIÓN slot\_display QUE ESTABLECE EL FORMATO EN EL QUE QUIERO IMPRIMIR LOS DATOS  
slot\_display <- function(prize){  
 symbols <- attr(prize, "symbols")  
 symbols <- paste(symbols, collapse = " ")  
 string <- paste(symbols, prize, sep = "\n$")  
cat(string)  
}  
  
  
#cREO Y A SU VEZ ASIGNO LA CLASE SLOTS A LA VARIABLE O FUNCIÓN one\_play  
class(one\_play) <- "slots" #Asigna la clase "slots" a la funcion one\_play  
  
#ASIGNO EL FORMATO DE IMPRESIÓN A LA CLASE SLOTS. TODO OBJETO QUE POSEA ESA CLASE SE IMPRIMIRÁ CON EL MIMSO FORMATO   
print.slots <- function(x, ...) {  
 slot\_display(x)  
}  
  
#A PARTIR DE ESTE PUNTO A TODOS LOS ELEMENTOS QUE SE LE ASIGENEN LA CLASE SLOTS SE IMPRIMIRAN BAJO ESE MISMO FORMATO ESTABLECIDO EN LA VARIABLE LÍNEAS ARRIBA POR EJEMPLO CON LA FUNCIÓN PLAY()  
play <- function() {  
 symbols <- get\_symbols()  
 structure(score(symbols), symbols = symbols, class = "slots")  
}

### Clases

1. Choose a name for your class.
2. Assign each instance of your class a class attribute.
3. Write class methods for any generic function likely to use objects of your class.

Puedo ver los metodos de una clase de la siguiente forma

methods(class = "factor")

## [1] [ [[ [[<- [<- all.equal   
## [6] as.character as.data.frame as.Date as.list as.logical   
## [11] as.POSIXlt as.vector c coerce droplevels   
## [16] format initialize is.na<- length<- levels<-   
## [21] Math Ops plot print relevel   
## [26] relist rep show slotsFromS3 summary   
## [31] Summary xtfrm   
## see '?methods' for accessing help and source code

La forma en la que afecta no usar metodos es la siguiente

play1 <- play()  
play1

## B BB BB  
## $5

play2 <- play()  
play2

## BBB 0 0  
## $0

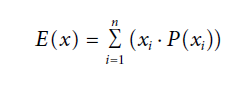
## 0 B 0 Ambas me imprimen información como esta porque son de la misma clase. Una vez las expreso como vectores cambia la clase por lo que a su vez cambia el metodo  
  
c(play1, play2)

## [1] 5 0

## [1] 5 0 Termina imprimiendo este valor ya que ha cambiado el métofo. E n este caso ambos valores pierden su clase slots por lo que no se aplica el método que se construyó para esa clase

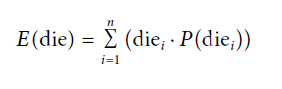
## Capitulo 9: loops

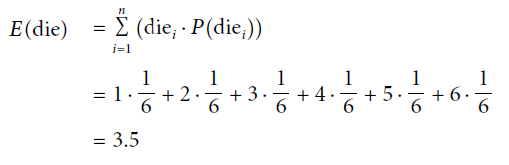
### Expected Values



die <- c(1, 2, 3, 4, 5, 6)

La fórmula para la probabilidad en ese caso sería





De igual forma podemos modificar los valores como se hizo en capítulos anteriores

### Expand.grid

Provee una vía rápida para tener cualquier combinación de elementos en n vectores

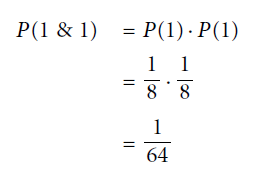
rolls <-expand.grid(die,die) #Me deveulve un objeto con el listado de todas las combinaciones posibles  
  
rolls <-expand.grid(die,die,die)  
  
rolls$value <- rolls$Var1 + rolls$Var2  
head(rolls,3)

## Var1 Var2 Var3 value  
## 1 1 1 1 2  
## 2 2 1 1 3  
## 3 3 1 1 4

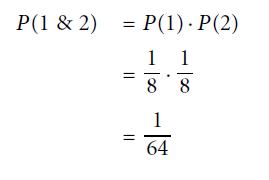
Ahora para calcular la probabilidad de toda esas combinaciones tenemos:  
*The probability that n independent, random events all occur is equal to the product of the probabilities that each random event occurs.*



So the probability that we roll a (1, 1) will be equal to the probability that we roll a one on the first die, 1/8, times the probability that we roll a one on the second die, 1/8:



And the probability that we roll a (1, 2) will be:



prob <- c("1" = 1/8, "2" = 1/8, "3" = 1/8, "4" = 1/8, "5" = 1/8, "6" = 3/8) #Establezco una probabilidad custom  
prob

## 1 2 3 4 5 6   
## 0.125 0.125 0.125 0.125 0.125 0.375

rolls$Var1

## [1] 1 2 3 4 5 6 1 2 3 4 5 6 1 2 3 4 5 6 1 2 3 4 5 6 1 2 3 4 5 6 1 2 3 4 5 6 1  
## [38] 2 3 4 5 6 1 2 3 4 5 6 1 2 3 4 5 6 1 2 3 4 5 6 1 2 3 4 5 6 1 2 3 4 5 6 1 2  
## [75] 3 4 5 6 1 2 3 4 5 6 1 2 3 4 5 6 1 2 3 4 5 6 1 2 3 4 5 6 1 2 3 4 5 6 1 2 3  
## [112] 4 5 6 1 2 3 4 5 6 1 2 3 4 5 6 1 2 3 4 5 6 1 2 3 4 5 6 1 2 3 4 5 6 1 2 3 4  
## [149] 5 6 1 2 3 4 5 6 1 2 3 4 5 6 1 2 3 4 5 6 1 2 3 4 5 6 1 2 3 4 5 6 1 2 3 4 5  
## [186] 6 1 2 3 4 5 6 1 2 3 4 5 6 1 2 3 4 5 6 1 2 3 4 5 6 1 2 3 4 5 6

prob[rolls$Var1] #Cambio las parobabilidades

## 1 2 3 4 5 6 1 2 3 4 5 6 1   
## 0.125 0.125 0.125 0.125 0.125 0.375 0.125 0.125 0.125 0.125 0.125 0.375 0.125   
## 2 3 4 5 6 1 2 3 4 5 6 1 2   
## 0.125 0.125 0.125 0.125 0.375 0.125 0.125 0.125 0.125 0.125 0.375 0.125 0.125   
## 3 4 5 6 1 2 3 4 5 6 1 2 3   
## 0.125 0.125 0.125 0.375 0.125 0.125 0.125 0.125 0.125 0.375 0.125 0.125 0.125   
## 4 5 6 1 2 3 4 5 6 1 2 3 4   
## 0.125 0.125 0.375 0.125 0.125 0.125 0.125 0.125 0.375 0.125 0.125 0.125 0.125   
## 5 6 1 2 3 4 5 6 1 2 3 4 5   
## 0.125 0.375 0.125 0.125 0.125 0.125 0.125 0.375 0.125 0.125 0.125 0.125 0.125   
## 6 1 2 3 4 5 6 1 2 3 4 5 6   
## 0.375 0.125 0.125 0.125 0.125 0.125 0.375 0.125 0.125 0.125 0.125 0.125 0.375   
## 1 2 3 4 5 6 1 2 3 4 5 6 1   
## 0.125 0.125 0.125 0.125 0.125 0.375 0.125 0.125 0.125 0.125 0.125 0.375 0.125   
## 2 3 4 5 6 1 2 3 4 5 6 1 2   
## 0.125 0.125 0.125 0.125 0.375 0.125 0.125 0.125 0.125 0.125 0.375 0.125 0.125   
## 3 4 5 6 1 2 3 4 5 6 1 2 3   
## 0.125 0.125 0.125 0.375 0.125 0.125 0.125 0.125 0.125 0.375 0.125 0.125 0.125   
## 4 5 6 1 2 3 4 5 6 1 2 3 4   
## 0.125 0.125 0.375 0.125 0.125 0.125 0.125 0.125 0.375 0.125 0.125 0.125 0.125   
## 5 6 1 2 3 4 5 6 1 2 3 4 5   
## 0.125 0.375 0.125 0.125 0.125 0.125 0.125 0.375 0.125 0.125 0.125 0.125 0.125   
## 6 1 2 3 4 5 6 1 2 3 4 5 6   
## 0.375 0.125 0.125 0.125 0.125 0.125 0.375 0.125 0.125 0.125 0.125 0.125 0.375   
## 1 2 3 4 5 6 1 2 3 4 5 6 1   
## 0.125 0.125 0.125 0.125 0.125 0.375 0.125 0.125 0.125 0.125 0.125 0.375 0.125   
## 2 3 4 5 6 1 2 3 4 5 6 1 2   
## 0.125 0.125 0.125 0.125 0.375 0.125 0.125 0.125 0.125 0.125 0.375 0.125 0.125   
## 3 4 5 6 1 2 3 4 5 6 1 2 3   
## 0.125 0.125 0.125 0.375 0.125 0.125 0.125 0.125 0.125 0.375 0.125 0.125 0.125   
## 4 5 6 1 2 3 4 5 6 1 2 3 4   
## 0.125 0.125 0.375 0.125 0.125 0.125 0.125 0.125 0.375 0.125 0.125 0.125 0.125   
## 5 6 1 2 3 4 5 6   
## 0.125 0.375 0.125 0.125 0.125 0.125 0.125 0.375

Ahora añado el valor de probabilidades como columna del dataframe de los dados

rolls$prob1 <- prob[rolls$Var1] #Cambio de probabilidades en el primer dado  
head(rolls, 3)

## Var1 Var2 Var3 value prob1  
## 1 1 1 1 2 0.125  
## 2 2 1 1 3 0.125  
## 3 3 1 1 4 0.125

rolls$prob2 <- prob[rolls$Var2]#Cambio de probabilidades en el segundo dado  
head(rolls, 3)

## Var1 Var2 Var3 value prob1 prob2  
## 1 1 1 1 2 0.125 0.125  
## 2 2 1 1 3 0.125 0.125  
## 3 3 1 1 4 0.125 0.125

rolls$prob <- rolls$prob1 \* rolls$prob2 #Cambio de probabilidades en el tercer dado  
head(rolls, 3)

## Var1 Var2 Var3 value prob1 prob2 prob  
## 1 1 1 1 2 0.125 0.125 0.015625  
## 2 2 1 1 3 0.125 0.125 0.015625  
## 3 3 1 1 4 0.125 0.125 0.015625

de acuerdo a las formulas entonces

sum(rolls$value \* rolls$prob)

## [1] 49.5

#### Ejercicio

Use expand.grid to make a data frame that contains every possible combination of three symbols from the wheel vector:

wheel <- c("DD", "7", "BBB", "BB", "B", "C", "0")  
  
combos <- expand.grid(wheel, wheel, wheel, stringsAsFactors = FALSE) #EL false es para evitar que genere los valores como factores  
  
combos

## Var1 Var2 Var3  
## 1 DD DD DD  
## 2 7 DD DD  
## 3 BBB DD DD  
## 4 BB DD DD  
## 5 B DD DD  
## 6 C DD DD  
## 7 0 DD DD  
## 8 DD 7 DD  
## 9 7 7 DD  
## 10 BBB 7 DD  
## 11 BB 7 DD  
## 12 B 7 DD  
## 13 C 7 DD  
## 14 0 7 DD  
## 15 DD BBB DD  
## 16 7 BBB DD  
## 17 BBB BBB DD  
## 18 BB BBB DD  
## 19 B BBB DD  
## 20 C BBB DD  
## 21 0 BBB DD  
## 22 DD BB DD  
## 23 7 BB DD  
## 24 BBB BB DD  
## 25 BB BB DD  
## 26 B BB DD  
## 27 C BB DD  
## 28 0 BB DD  
## 29 DD B DD  
## 30 7 B DD  
## 31 BBB B DD  
## 32 BB B DD  
## 33 B B DD  
## 34 C B DD  
## 35 0 B DD  
## 36 DD C DD  
## 37 7 C DD  
## 38 BBB C DD  
## 39 BB C DD  
## 40 B C DD  
## 41 C C DD  
## 42 0 C DD  
## 43 DD 0 DD  
## 44 7 0 DD  
## 45 BBB 0 DD  
## 46 BB 0 DD  
## 47 B 0 DD  
## 48 C 0 DD  
## 49 0 0 DD  
## 50 DD DD 7  
## 51 7 DD 7  
## 52 BBB DD 7  
## 53 BB DD 7  
## 54 B DD 7  
## 55 C DD 7  
## 56 0 DD 7  
## 57 DD 7 7  
## 58 7 7 7  
## 59 BBB 7 7  
## 60 BB 7 7  
## 61 B 7 7  
## 62 C 7 7  
## 63 0 7 7  
## 64 DD BBB 7  
## 65 7 BBB 7  
## 66 BBB BBB 7  
## 67 BB BBB 7  
## 68 B BBB 7  
## 69 C BBB 7  
## 70 0 BBB 7  
## 71 DD BB 7  
## 72 7 BB 7  
## 73 BBB BB 7  
## 74 BB BB 7  
## 75 B BB 7  
## 76 C BB 7  
## 77 0 BB 7  
## 78 DD B 7  
## 79 7 B 7  
## 80 BBB B 7  
## 81 BB B 7  
## 82 B B 7  
## 83 C B 7  
## 84 0 B 7  
## 85 DD C 7  
## 86 7 C 7  
## 87 BBB C 7  
## 88 BB C 7  
## 89 B C 7  
## 90 C C 7  
## 91 0 C 7  
## 92 DD 0 7  
## 93 7 0 7  
## 94 BBB 0 7  
## 95 BB 0 7  
## 96 B 0 7  
## 97 C 0 7  
## 98 0 0 7  
## 99 DD DD BBB  
## 100 7 DD BBB  
## 101 BBB DD BBB  
## 102 BB DD BBB  
## 103 B DD BBB  
## 104 C DD BBB  
## 105 0 DD BBB  
## 106 DD 7 BBB  
## 107 7 7 BBB  
## 108 BBB 7 BBB  
## 109 BB 7 BBB  
## 110 B 7 BBB  
## 111 C 7 BBB  
## 112 0 7 BBB  
## 113 DD BBB BBB  
## 114 7 BBB BBB  
## 115 BBB BBB BBB  
## 116 BB BBB BBB  
## 117 B BBB BBB  
## 118 C BBB BBB  
## 119 0 BBB BBB  
## 120 DD BB BBB  
## 121 7 BB BBB  
## 122 BBB BB BBB  
## 123 BB BB BBB  
## 124 B BB BBB  
## 125 C BB BBB  
## 126 0 BB BBB  
## 127 DD B BBB  
## 128 7 B BBB  
## 129 BBB B BBB  
## 130 BB B BBB  
## 131 B B BBB  
## 132 C B BBB  
## 133 0 B BBB  
## 134 DD C BBB  
## 135 7 C BBB  
## 136 BBB C BBB  
## 137 BB C BBB  
## 138 B C BBB  
## 139 C C BBB  
## 140 0 C BBB  
## 141 DD 0 BBB  
## 142 7 0 BBB  
## 143 BBB 0 BBB  
## 144 BB 0 BBB  
## 145 B 0 BBB  
## 146 C 0 BBB  
## 147 0 0 BBB  
## 148 DD DD BB  
## 149 7 DD BB  
## 150 BBB DD BB  
## 151 BB DD BB  
## 152 B DD BB  
## 153 C DD BB  
## 154 0 DD BB  
## 155 DD 7 BB  
## 156 7 7 BB  
## 157 BBB 7 BB  
## 158 BB 7 BB  
## 159 B 7 BB  
## 160 C 7 BB  
## 161 0 7 BB  
## 162 DD BBB BB  
## 163 7 BBB BB  
## 164 BBB BBB BB  
## 165 BB BBB BB  
## 166 B BBB BB  
## 167 C BBB BB  
## 168 0 BBB BB  
## 169 DD BB BB  
## 170 7 BB BB  
## 171 BBB BB BB  
## 172 BB BB BB  
## 173 B BB BB  
## 174 C BB BB  
## 175 0 BB BB  
## 176 DD B BB  
## 177 7 B BB  
## 178 BBB B BB  
## 179 BB B BB  
## 180 B B BB  
## 181 C B BB  
## 182 0 B BB  
## 183 DD C BB  
## 184 7 C BB  
## 185 BBB C BB  
## 186 BB C BB  
## 187 B C BB  
## 188 C C BB  
## 189 0 C BB  
## 190 DD 0 BB  
## 191 7 0 BB  
## 192 BBB 0 BB  
## 193 BB 0 BB  
## 194 B 0 BB  
## 195 C 0 BB  
## 196 0 0 BB  
## 197 DD DD B  
## 198 7 DD B  
## 199 BBB DD B  
## 200 BB DD B  
## 201 B DD B  
## 202 C DD B  
## 203 0 DD B  
## 204 DD 7 B  
## 205 7 7 B  
## 206 BBB 7 B  
## 207 BB 7 B  
## 208 B 7 B  
## 209 C 7 B  
## 210 0 7 B  
## 211 DD BBB B  
## 212 7 BBB B  
## 213 BBB BBB B  
## 214 BB BBB B  
## 215 B BBB B  
## 216 C BBB B  
## 217 0 BBB B  
## 218 DD BB B  
## 219 7 BB B  
## 220 BBB BB B  
## 221 BB BB B  
## 222 B BB B  
## 223 C BB B  
## 224 0 BB B  
## 225 DD B B  
## 226 7 B B  
## 227 BBB B B  
## 228 BB B B  
## 229 B B B  
## 230 C B B  
## 231 0 B B  
## 232 DD C B  
## 233 7 C B  
## 234 BBB C B  
## 235 BB C B  
## 236 B C B  
## 237 C C B  
## 238 0 C B  
## 239 DD 0 B  
## 240 7 0 B  
## 241 BBB 0 B  
## 242 BB 0 B  
## 243 B 0 B  
## 244 C 0 B  
## 245 0 0 B  
## 246 DD DD C  
## 247 7 DD C  
## 248 BBB DD C  
## 249 BB DD C  
## 250 B DD C  
## 251 C DD C  
## 252 0 DD C  
## 253 DD 7 C  
## 254 7 7 C  
## 255 BBB 7 C  
## 256 BB 7 C  
## 257 B 7 C  
## 258 C 7 C  
## 259 0 7 C  
## 260 DD BBB C  
## 261 7 BBB C  
## 262 BBB BBB C  
## 263 BB BBB C  
## 264 B BBB C  
## 265 C BBB C  
## 266 0 BBB C  
## 267 DD BB C  
## 268 7 BB C  
## 269 BBB BB C  
## 270 BB BB C  
## 271 B BB C  
## 272 C BB C  
## 273 0 BB C  
## 274 DD B C  
## 275 7 B C  
## 276 BBB B C  
## 277 BB B C  
## 278 B B C  
## 279 C B C  
## 280 0 B C  
## 281 DD C C  
## 282 7 C C  
## 283 BBB C C  
## 284 BB C C  
## 285 B C C  
## 286 C C C  
## 287 0 C C  
## 288 DD 0 C  
## 289 7 0 C  
## 290 BBB 0 C  
## 291 BB 0 C  
## 292 B 0 C  
## 293 C 0 C  
## 294 0 0 C  
## 295 DD DD 0  
## 296 7 DD 0  
## 297 BBB DD 0  
## 298 BB DD 0  
## 299 B DD 0  
## 300 C DD 0  
## 301 0 DD 0  
## 302 DD 7 0  
## 303 7 7 0  
## 304 BBB 7 0  
## 305 BB 7 0  
## 306 B 7 0  
## 307 C 7 0  
## 308 0 7 0  
## 309 DD BBB 0  
## 310 7 BBB 0  
## 311 BBB BBB 0  
## 312 BB BBB 0  
## 313 B BBB 0  
## 314 C BBB 0  
## 315 0 BBB 0  
## 316 DD BB 0  
## 317 7 BB 0  
## 318 BBB BB 0  
## 319 BB BB 0  
## 320 B BB 0  
## 321 C BB 0  
## 322 0 BB 0  
## 323 DD B 0  
## 324 7 B 0  
## 325 BBB B 0  
## 326 BB B 0  
## 327 B B 0  
## 328 C B 0  
## 329 0 B 0  
## 330 DD C 0  
## 331 7 C 0  
## 332 BBB C 0  
## 333 BB C 0  
## 334 B C 0  
## 335 C C 0  
## 336 0 C 0  
## 337 DD 0 0  
## 338 7 0 0  
## 339 BBB 0 0  
## 340 BB 0 0  
## 341 B 0 0  
## 342 C 0 0  
## 343 0 0 0

Ahora calculamos la probabilidad de obtener cada combinación. Usamos las que generamos para la slot machine

get\_symbols <- function() {  
 wheel <- c("DD", "7", "BBB", "BB", "B", "C", "0")  
 sample(wheel, size = 3, replace = TRUE) #Obtiene una muesra de tres datos entre todos los daos del objeto wheel  
 prob = c(0.03, 0.03, 0.06, 0.1, 0.25, 0.01, 0.52)  
}

O también podemos generarlas a través de una look up table

prob <- c("DD" = 0.03, "7" = 0.03, "BBB" = 0.06, "BB" = 0.1, "B" = 0.25, "C" = 0.01, "0" = 0.52)  
  
#se genera columnas con las probablidades  
combos$prob1 <- prob[combos$Var1]  
combos$prob2 <- prob[combos$Var2]  
combos$prob3 <- prob[combos$Var3]  
head(combos, 3)

## Var1 Var2 Var3 prob1 prob2 prob3  
## 1 DD DD DD 0.03 0.03 0.03  
## 2 7 DD DD 0.03 0.03 0.03  
## 3 BBB DD DD 0.06 0.03 0.03

Ahora calculamos la probabilidad total de cada combinación

combos$prob <- combos$prob1 \* combos$prob2 \* combos$prob3  
head(combos, 3)

## Var1 Var2 Var3 prob1 prob2 prob3 prob  
## 1 DD DD DD 0.03 0.03 0.03 2.7e-05  
## 2 7 DD DD 0.03 0.03 0.03 2.7e-05  
## 3 BBB DD DD 0.06 0.03 0.03 5.4e-05

sum(combos$prob)

## [1] 1

Puedo calcular el precio de cada combinación de la siguiente forma

symbols <- c(combos[1, 1], combos[1, 2], combos[1, 3])  
symbols

## [1] "DD" "DD" "DD"

score(symbols)

## [1] 800

Pero tenemos 343 combinaciones lo cual hacerlo uno a uno es una tarea imposible. Ahí es donde entran los loops

### for loops

for (value in that) {

this

}

for (value in c("My", "first", "for", "loop")) { #Hay tres elementos por lo que se ejecuta tres veces la acción descrita más adelante  
 print("one run")  
}

## [1] "one run"  
## [1] "one run"  
## [1] "one run"  
## [1] "one run"

Si quiere iprimir cada valor del vector que está dentro del vector debo imprimir dicho valor de la suguiente forma

for (value in c("My", "second", "for", "loop")) {  
 print(value)  
}

## [1] "My"  
## [1] "second"  
## [1] "for"  
## [1] "loop"

No es necesario usar value como primer argumento

for (word in c("My", "second", "for", "loop")) {  
 print(word)  
}

## [1] "My"  
## [1] "second"  
## [1] "for"  
## [1] "loop"

for (string in c("My", "second", "for", "loop")) {  
 print(string)  
}

## [1] "My"  
## [1] "second"  
## [1] "for"  
## [1] "loop"

for (i in c("My", "second", "for", "loop")) {  
 print(i)  
}

## [1] "My"  
## [1] "second"  
## [1] "for"  
## [1] "loop"

Para guardar los datos en una variable

chars <- vector(length = 4)  
  
words <- c("My", "fourth", "for", "loop")  
 for (i in 1:4) {  
 chars[i] <- words[i]  
}  
chars

## [1] "My" "fourth" "for" "loop"

Let’s use a for loop to calculate the prize for each row in combos

combos$prize <- NA  
combos

## Var1 Var2 Var3 prob1 prob2 prob3 prob prize  
## 1 DD DD DD 0.03 0.03 0.03 0.000027 NA  
## 2 7 DD DD 0.03 0.03 0.03 0.000027 NA  
## 3 BBB DD DD 0.06 0.03 0.03 0.000054 NA  
## 4 BB DD DD 0.10 0.03 0.03 0.000090 NA  
## 5 B DD DD 0.25 0.03 0.03 0.000225 NA  
## 6 C DD DD 0.01 0.03 0.03 0.000009 NA  
## 7 0 DD DD 0.52 0.03 0.03 0.000468 NA  
## 8 DD 7 DD 0.03 0.03 0.03 0.000027 NA  
## 9 7 7 DD 0.03 0.03 0.03 0.000027 NA  
## 10 BBB 7 DD 0.06 0.03 0.03 0.000054 NA  
## 11 BB 7 DD 0.10 0.03 0.03 0.000090 NA  
## 12 B 7 DD 0.25 0.03 0.03 0.000225 NA  
## 13 C 7 DD 0.01 0.03 0.03 0.000009 NA  
## 14 0 7 DD 0.52 0.03 0.03 0.000468 NA  
## 15 DD BBB DD 0.03 0.06 0.03 0.000054 NA  
## 16 7 BBB DD 0.03 0.06 0.03 0.000054 NA  
## 17 BBB BBB DD 0.06 0.06 0.03 0.000108 NA  
## 18 BB BBB DD 0.10 0.06 0.03 0.000180 NA  
## 19 B BBB DD 0.25 0.06 0.03 0.000450 NA  
## 20 C BBB DD 0.01 0.06 0.03 0.000018 NA  
## 21 0 BBB DD 0.52 0.06 0.03 0.000936 NA  
## 22 DD BB DD 0.03 0.10 0.03 0.000090 NA  
## 23 7 BB DD 0.03 0.10 0.03 0.000090 NA  
## 24 BBB BB DD 0.06 0.10 0.03 0.000180 NA  
## 25 BB BB DD 0.10 0.10 0.03 0.000300 NA  
## 26 B BB DD 0.25 0.10 0.03 0.000750 NA  
## 27 C BB DD 0.01 0.10 0.03 0.000030 NA  
## 28 0 BB DD 0.52 0.10 0.03 0.001560 NA  
## 29 DD B DD 0.03 0.25 0.03 0.000225 NA  
## 30 7 B DD 0.03 0.25 0.03 0.000225 NA  
## 31 BBB B DD 0.06 0.25 0.03 0.000450 NA  
## 32 BB B DD 0.10 0.25 0.03 0.000750 NA  
## 33 B B DD 0.25 0.25 0.03 0.001875 NA  
## 34 C B DD 0.01 0.25 0.03 0.000075 NA  
## 35 0 B DD 0.52 0.25 0.03 0.003900 NA  
## 36 DD C DD 0.03 0.01 0.03 0.000009 NA  
## 37 7 C DD 0.03 0.01 0.03 0.000009 NA  
## 38 BBB C DD 0.06 0.01 0.03 0.000018 NA  
## 39 BB C DD 0.10 0.01 0.03 0.000030 NA  
## 40 B C DD 0.25 0.01 0.03 0.000075 NA  
## 41 C C DD 0.01 0.01 0.03 0.000003 NA  
## 42 0 C DD 0.52 0.01 0.03 0.000156 NA  
## 43 DD 0 DD 0.03 0.52 0.03 0.000468 NA  
## 44 7 0 DD 0.03 0.52 0.03 0.000468 NA  
## 45 BBB 0 DD 0.06 0.52 0.03 0.000936 NA  
## 46 BB 0 DD 0.10 0.52 0.03 0.001560 NA  
## 47 B 0 DD 0.25 0.52 0.03 0.003900 NA  
## 48 C 0 DD 0.01 0.52 0.03 0.000156 NA  
## 49 0 0 DD 0.52 0.52 0.03 0.008112 NA  
## 50 DD DD 7 0.03 0.03 0.03 0.000027 NA  
## 51 7 DD 7 0.03 0.03 0.03 0.000027 NA  
## 52 BBB DD 7 0.06 0.03 0.03 0.000054 NA  
## 53 BB DD 7 0.10 0.03 0.03 0.000090 NA  
## 54 B DD 7 0.25 0.03 0.03 0.000225 NA  
## 55 C DD 7 0.01 0.03 0.03 0.000009 NA  
## 56 0 DD 7 0.52 0.03 0.03 0.000468 NA  
## 57 DD 7 7 0.03 0.03 0.03 0.000027 NA  
## 58 7 7 7 0.03 0.03 0.03 0.000027 NA  
## 59 BBB 7 7 0.06 0.03 0.03 0.000054 NA  
## 60 BB 7 7 0.10 0.03 0.03 0.000090 NA  
## 61 B 7 7 0.25 0.03 0.03 0.000225 NA  
## 62 C 7 7 0.01 0.03 0.03 0.000009 NA  
## 63 0 7 7 0.52 0.03 0.03 0.000468 NA  
## 64 DD BBB 7 0.03 0.06 0.03 0.000054 NA  
## 65 7 BBB 7 0.03 0.06 0.03 0.000054 NA  
## 66 BBB BBB 7 0.06 0.06 0.03 0.000108 NA  
## 67 BB BBB 7 0.10 0.06 0.03 0.000180 NA  
## 68 B BBB 7 0.25 0.06 0.03 0.000450 NA  
## 69 C BBB 7 0.01 0.06 0.03 0.000018 NA  
## 70 0 BBB 7 0.52 0.06 0.03 0.000936 NA  
## 71 DD BB 7 0.03 0.10 0.03 0.000090 NA  
## 72 7 BB 7 0.03 0.10 0.03 0.000090 NA  
## 73 BBB BB 7 0.06 0.10 0.03 0.000180 NA  
## 74 BB BB 7 0.10 0.10 0.03 0.000300 NA  
## 75 B BB 7 0.25 0.10 0.03 0.000750 NA  
## 76 C BB 7 0.01 0.10 0.03 0.000030 NA  
## 77 0 BB 7 0.52 0.10 0.03 0.001560 NA  
## 78 DD B 7 0.03 0.25 0.03 0.000225 NA  
## 79 7 B 7 0.03 0.25 0.03 0.000225 NA  
## 80 BBB B 7 0.06 0.25 0.03 0.000450 NA  
## 81 BB B 7 0.10 0.25 0.03 0.000750 NA  
## 82 B B 7 0.25 0.25 0.03 0.001875 NA  
## 83 C B 7 0.01 0.25 0.03 0.000075 NA  
## 84 0 B 7 0.52 0.25 0.03 0.003900 NA  
## 85 DD C 7 0.03 0.01 0.03 0.000009 NA  
## 86 7 C 7 0.03 0.01 0.03 0.000009 NA  
## 87 BBB C 7 0.06 0.01 0.03 0.000018 NA  
## 88 BB C 7 0.10 0.01 0.03 0.000030 NA  
## 89 B C 7 0.25 0.01 0.03 0.000075 NA  
## 90 C C 7 0.01 0.01 0.03 0.000003 NA  
## 91 0 C 7 0.52 0.01 0.03 0.000156 NA  
## 92 DD 0 7 0.03 0.52 0.03 0.000468 NA  
## 93 7 0 7 0.03 0.52 0.03 0.000468 NA  
## 94 BBB 0 7 0.06 0.52 0.03 0.000936 NA  
## 95 BB 0 7 0.10 0.52 0.03 0.001560 NA  
## 96 B 0 7 0.25 0.52 0.03 0.003900 NA  
## 97 C 0 7 0.01 0.52 0.03 0.000156 NA  
## 98 0 0 7 0.52 0.52 0.03 0.008112 NA  
## 99 DD DD BBB 0.03 0.03 0.06 0.000054 NA  
## 100 7 DD BBB 0.03 0.03 0.06 0.000054 NA  
## 101 BBB DD BBB 0.06 0.03 0.06 0.000108 NA  
## 102 BB DD BBB 0.10 0.03 0.06 0.000180 NA  
## 103 B DD BBB 0.25 0.03 0.06 0.000450 NA  
## 104 C DD BBB 0.01 0.03 0.06 0.000018 NA  
## 105 0 DD BBB 0.52 0.03 0.06 0.000936 NA  
## 106 DD 7 BBB 0.03 0.03 0.06 0.000054 NA  
## 107 7 7 BBB 0.03 0.03 0.06 0.000054 NA  
## 108 BBB 7 BBB 0.06 0.03 0.06 0.000108 NA  
## 109 BB 7 BBB 0.10 0.03 0.06 0.000180 NA  
## 110 B 7 BBB 0.25 0.03 0.06 0.000450 NA  
## 111 C 7 BBB 0.01 0.03 0.06 0.000018 NA  
## 112 0 7 BBB 0.52 0.03 0.06 0.000936 NA  
## 113 DD BBB BBB 0.03 0.06 0.06 0.000108 NA  
## 114 7 BBB BBB 0.03 0.06 0.06 0.000108 NA  
## 115 BBB BBB BBB 0.06 0.06 0.06 0.000216 NA  
## 116 BB BBB BBB 0.10 0.06 0.06 0.000360 NA  
## 117 B BBB BBB 0.25 0.06 0.06 0.000900 NA  
## 118 C BBB BBB 0.01 0.06 0.06 0.000036 NA  
## 119 0 BBB BBB 0.52 0.06 0.06 0.001872 NA  
## 120 DD BB BBB 0.03 0.10 0.06 0.000180 NA  
## 121 7 BB BBB 0.03 0.10 0.06 0.000180 NA  
## 122 BBB BB BBB 0.06 0.10 0.06 0.000360 NA  
## 123 BB BB BBB 0.10 0.10 0.06 0.000600 NA  
## 124 B BB BBB 0.25 0.10 0.06 0.001500 NA  
## 125 C BB BBB 0.01 0.10 0.06 0.000060 NA  
## 126 0 BB BBB 0.52 0.10 0.06 0.003120 NA  
## 127 DD B BBB 0.03 0.25 0.06 0.000450 NA  
## 128 7 B BBB 0.03 0.25 0.06 0.000450 NA  
## 129 BBB B BBB 0.06 0.25 0.06 0.000900 NA  
## 130 BB B BBB 0.10 0.25 0.06 0.001500 NA  
## 131 B B BBB 0.25 0.25 0.06 0.003750 NA  
## 132 C B BBB 0.01 0.25 0.06 0.000150 NA  
## 133 0 B BBB 0.52 0.25 0.06 0.007800 NA  
## 134 DD C BBB 0.03 0.01 0.06 0.000018 NA  
## 135 7 C BBB 0.03 0.01 0.06 0.000018 NA  
## 136 BBB C BBB 0.06 0.01 0.06 0.000036 NA  
## 137 BB C BBB 0.10 0.01 0.06 0.000060 NA  
## 138 B C BBB 0.25 0.01 0.06 0.000150 NA  
## 139 C C BBB 0.01 0.01 0.06 0.000006 NA  
## 140 0 C BBB 0.52 0.01 0.06 0.000312 NA  
## 141 DD 0 BBB 0.03 0.52 0.06 0.000936 NA  
## 142 7 0 BBB 0.03 0.52 0.06 0.000936 NA  
## 143 BBB 0 BBB 0.06 0.52 0.06 0.001872 NA  
## 144 BB 0 BBB 0.10 0.52 0.06 0.003120 NA  
## 145 B 0 BBB 0.25 0.52 0.06 0.007800 NA  
## 146 C 0 BBB 0.01 0.52 0.06 0.000312 NA  
## 147 0 0 BBB 0.52 0.52 0.06 0.016224 NA  
## 148 DD DD BB 0.03 0.03 0.10 0.000090 NA  
## 149 7 DD BB 0.03 0.03 0.10 0.000090 NA  
## 150 BBB DD BB 0.06 0.03 0.10 0.000180 NA  
## 151 BB DD BB 0.10 0.03 0.10 0.000300 NA  
## 152 B DD BB 0.25 0.03 0.10 0.000750 NA  
## 153 C DD BB 0.01 0.03 0.10 0.000030 NA  
## 154 0 DD BB 0.52 0.03 0.10 0.001560 NA  
## 155 DD 7 BB 0.03 0.03 0.10 0.000090 NA  
## 156 7 7 BB 0.03 0.03 0.10 0.000090 NA  
## 157 BBB 7 BB 0.06 0.03 0.10 0.000180 NA  
## 158 BB 7 BB 0.10 0.03 0.10 0.000300 NA  
## 159 B 7 BB 0.25 0.03 0.10 0.000750 NA  
## 160 C 7 BB 0.01 0.03 0.10 0.000030 NA  
## 161 0 7 BB 0.52 0.03 0.10 0.001560 NA  
## 162 DD BBB BB 0.03 0.06 0.10 0.000180 NA  
## 163 7 BBB BB 0.03 0.06 0.10 0.000180 NA  
## 164 BBB BBB BB 0.06 0.06 0.10 0.000360 NA  
## 165 BB BBB BB 0.10 0.06 0.10 0.000600 NA  
## 166 B BBB BB 0.25 0.06 0.10 0.001500 NA  
## 167 C BBB BB 0.01 0.06 0.10 0.000060 NA  
## 168 0 BBB BB 0.52 0.06 0.10 0.003120 NA  
## 169 DD BB BB 0.03 0.10 0.10 0.000300 NA  
## 170 7 BB BB 0.03 0.10 0.10 0.000300 NA  
## 171 BBB BB BB 0.06 0.10 0.10 0.000600 NA  
## 172 BB BB BB 0.10 0.10 0.10 0.001000 NA  
## 173 B BB BB 0.25 0.10 0.10 0.002500 NA  
## 174 C BB BB 0.01 0.10 0.10 0.000100 NA  
## 175 0 BB BB 0.52 0.10 0.10 0.005200 NA  
## 176 DD B BB 0.03 0.25 0.10 0.000750 NA  
## 177 7 B BB 0.03 0.25 0.10 0.000750 NA  
## 178 BBB B BB 0.06 0.25 0.10 0.001500 NA  
## 179 BB B BB 0.10 0.25 0.10 0.002500 NA  
## 180 B B BB 0.25 0.25 0.10 0.006250 NA  
## 181 C B BB 0.01 0.25 0.10 0.000250 NA  
## 182 0 B BB 0.52 0.25 0.10 0.013000 NA  
## 183 DD C BB 0.03 0.01 0.10 0.000030 NA  
## 184 7 C BB 0.03 0.01 0.10 0.000030 NA  
## 185 BBB C BB 0.06 0.01 0.10 0.000060 NA  
## 186 BB C BB 0.10 0.01 0.10 0.000100 NA  
## 187 B C BB 0.25 0.01 0.10 0.000250 NA  
## 188 C C BB 0.01 0.01 0.10 0.000010 NA  
## 189 0 C BB 0.52 0.01 0.10 0.000520 NA  
## 190 DD 0 BB 0.03 0.52 0.10 0.001560 NA  
## 191 7 0 BB 0.03 0.52 0.10 0.001560 NA  
## 192 BBB 0 BB 0.06 0.52 0.10 0.003120 NA  
## 193 BB 0 BB 0.10 0.52 0.10 0.005200 NA  
## 194 B 0 BB 0.25 0.52 0.10 0.013000 NA  
## 195 C 0 BB 0.01 0.52 0.10 0.000520 NA  
## 196 0 0 BB 0.52 0.52 0.10 0.027040 NA  
## 197 DD DD B 0.03 0.03 0.25 0.000225 NA  
## 198 7 DD B 0.03 0.03 0.25 0.000225 NA  
## 199 BBB DD B 0.06 0.03 0.25 0.000450 NA  
## 200 BB DD B 0.10 0.03 0.25 0.000750 NA  
## 201 B DD B 0.25 0.03 0.25 0.001875 NA  
## 202 C DD B 0.01 0.03 0.25 0.000075 NA  
## 203 0 DD B 0.52 0.03 0.25 0.003900 NA  
## 204 DD 7 B 0.03 0.03 0.25 0.000225 NA  
## 205 7 7 B 0.03 0.03 0.25 0.000225 NA  
## 206 BBB 7 B 0.06 0.03 0.25 0.000450 NA  
## 207 BB 7 B 0.10 0.03 0.25 0.000750 NA  
## 208 B 7 B 0.25 0.03 0.25 0.001875 NA  
## 209 C 7 B 0.01 0.03 0.25 0.000075 NA  
## 210 0 7 B 0.52 0.03 0.25 0.003900 NA  
## 211 DD BBB B 0.03 0.06 0.25 0.000450 NA  
## 212 7 BBB B 0.03 0.06 0.25 0.000450 NA  
## 213 BBB BBB B 0.06 0.06 0.25 0.000900 NA  
## 214 BB BBB B 0.10 0.06 0.25 0.001500 NA  
## 215 B BBB B 0.25 0.06 0.25 0.003750 NA  
## 216 C BBB B 0.01 0.06 0.25 0.000150 NA  
## 217 0 BBB B 0.52 0.06 0.25 0.007800 NA  
## 218 DD BB B 0.03 0.10 0.25 0.000750 NA  
## 219 7 BB B 0.03 0.10 0.25 0.000750 NA  
## 220 BBB BB B 0.06 0.10 0.25 0.001500 NA  
## 221 BB BB B 0.10 0.10 0.25 0.002500 NA  
## 222 B BB B 0.25 0.10 0.25 0.006250 NA  
## 223 C BB B 0.01 0.10 0.25 0.000250 NA  
## 224 0 BB B 0.52 0.10 0.25 0.013000 NA  
## 225 DD B B 0.03 0.25 0.25 0.001875 NA  
## 226 7 B B 0.03 0.25 0.25 0.001875 NA  
## 227 BBB B B 0.06 0.25 0.25 0.003750 NA  
## 228 BB B B 0.10 0.25 0.25 0.006250 NA  
## 229 B B B 0.25 0.25 0.25 0.015625 NA  
## 230 C B B 0.01 0.25 0.25 0.000625 NA  
## 231 0 B B 0.52 0.25 0.25 0.032500 NA  
## 232 DD C B 0.03 0.01 0.25 0.000075 NA  
## 233 7 C B 0.03 0.01 0.25 0.000075 NA  
## 234 BBB C B 0.06 0.01 0.25 0.000150 NA  
## 235 BB C B 0.10 0.01 0.25 0.000250 NA  
## 236 B C B 0.25 0.01 0.25 0.000625 NA  
## 237 C C B 0.01 0.01 0.25 0.000025 NA  
## 238 0 C B 0.52 0.01 0.25 0.001300 NA  
## 239 DD 0 B 0.03 0.52 0.25 0.003900 NA  
## 240 7 0 B 0.03 0.52 0.25 0.003900 NA  
## 241 BBB 0 B 0.06 0.52 0.25 0.007800 NA  
## 242 BB 0 B 0.10 0.52 0.25 0.013000 NA  
## 243 B 0 B 0.25 0.52 0.25 0.032500 NA  
## 244 C 0 B 0.01 0.52 0.25 0.001300 NA  
## 245 0 0 B 0.52 0.52 0.25 0.067600 NA  
## 246 DD DD C 0.03 0.03 0.01 0.000009 NA  
## 247 7 DD C 0.03 0.03 0.01 0.000009 NA  
## 248 BBB DD C 0.06 0.03 0.01 0.000018 NA  
## 249 BB DD C 0.10 0.03 0.01 0.000030 NA  
## 250 B DD C 0.25 0.03 0.01 0.000075 NA  
## 251 C DD C 0.01 0.03 0.01 0.000003 NA  
## 252 0 DD C 0.52 0.03 0.01 0.000156 NA  
## 253 DD 7 C 0.03 0.03 0.01 0.000009 NA  
## 254 7 7 C 0.03 0.03 0.01 0.000009 NA  
## 255 BBB 7 C 0.06 0.03 0.01 0.000018 NA  
## 256 BB 7 C 0.10 0.03 0.01 0.000030 NA  
## 257 B 7 C 0.25 0.03 0.01 0.000075 NA  
## 258 C 7 C 0.01 0.03 0.01 0.000003 NA  
## 259 0 7 C 0.52 0.03 0.01 0.000156 NA  
## 260 DD BBB C 0.03 0.06 0.01 0.000018 NA  
## 261 7 BBB C 0.03 0.06 0.01 0.000018 NA  
## 262 BBB BBB C 0.06 0.06 0.01 0.000036 NA  
## 263 BB BBB C 0.10 0.06 0.01 0.000060 NA  
## 264 B BBB C 0.25 0.06 0.01 0.000150 NA  
## 265 C BBB C 0.01 0.06 0.01 0.000006 NA  
## 266 0 BBB C 0.52 0.06 0.01 0.000312 NA  
## 267 DD BB C 0.03 0.10 0.01 0.000030 NA  
## 268 7 BB C 0.03 0.10 0.01 0.000030 NA  
## 269 BBB BB C 0.06 0.10 0.01 0.000060 NA  
## 270 BB BB C 0.10 0.10 0.01 0.000100 NA  
## 271 B BB C 0.25 0.10 0.01 0.000250 NA  
## 272 C BB C 0.01 0.10 0.01 0.000010 NA  
## 273 0 BB C 0.52 0.10 0.01 0.000520 NA  
## 274 DD B C 0.03 0.25 0.01 0.000075 NA  
## 275 7 B C 0.03 0.25 0.01 0.000075 NA  
## 276 BBB B C 0.06 0.25 0.01 0.000150 NA  
## 277 BB B C 0.10 0.25 0.01 0.000250 NA  
## 278 B B C 0.25 0.25 0.01 0.000625 NA  
## 279 C B C 0.01 0.25 0.01 0.000025 NA  
## 280 0 B C 0.52 0.25 0.01 0.001300 NA  
## 281 DD C C 0.03 0.01 0.01 0.000003 NA  
## 282 7 C C 0.03 0.01 0.01 0.000003 NA  
## 283 BBB C C 0.06 0.01 0.01 0.000006 NA  
## 284 BB C C 0.10 0.01 0.01 0.000010 NA  
## 285 B C C 0.25 0.01 0.01 0.000025 NA  
## 286 C C C 0.01 0.01 0.01 0.000001 NA  
## 287 0 C C 0.52 0.01 0.01 0.000052 NA  
## 288 DD 0 C 0.03 0.52 0.01 0.000156 NA  
## 289 7 0 C 0.03 0.52 0.01 0.000156 NA  
## 290 BBB 0 C 0.06 0.52 0.01 0.000312 NA  
## 291 BB 0 C 0.10 0.52 0.01 0.000520 NA  
## 292 B 0 C 0.25 0.52 0.01 0.001300 NA  
## 293 C 0 C 0.01 0.52 0.01 0.000052 NA  
## 294 0 0 C 0.52 0.52 0.01 0.002704 NA  
## 295 DD DD 0 0.03 0.03 0.52 0.000468 NA  
## 296 7 DD 0 0.03 0.03 0.52 0.000468 NA  
## 297 BBB DD 0 0.06 0.03 0.52 0.000936 NA  
## 298 BB DD 0 0.10 0.03 0.52 0.001560 NA  
## 299 B DD 0 0.25 0.03 0.52 0.003900 NA  
## 300 C DD 0 0.01 0.03 0.52 0.000156 NA  
## 301 0 DD 0 0.52 0.03 0.52 0.008112 NA  
## 302 DD 7 0 0.03 0.03 0.52 0.000468 NA  
## 303 7 7 0 0.03 0.03 0.52 0.000468 NA  
## 304 BBB 7 0 0.06 0.03 0.52 0.000936 NA  
## 305 BB 7 0 0.10 0.03 0.52 0.001560 NA  
## 306 B 7 0 0.25 0.03 0.52 0.003900 NA  
## 307 C 7 0 0.01 0.03 0.52 0.000156 NA  
## 308 0 7 0 0.52 0.03 0.52 0.008112 NA  
## 309 DD BBB 0 0.03 0.06 0.52 0.000936 NA  
## 310 7 BBB 0 0.03 0.06 0.52 0.000936 NA  
## 311 BBB BBB 0 0.06 0.06 0.52 0.001872 NA  
## 312 BB BBB 0 0.10 0.06 0.52 0.003120 NA  
## 313 B BBB 0 0.25 0.06 0.52 0.007800 NA  
## 314 C BBB 0 0.01 0.06 0.52 0.000312 NA  
## 315 0 BBB 0 0.52 0.06 0.52 0.016224 NA  
## 316 DD BB 0 0.03 0.10 0.52 0.001560 NA  
## 317 7 BB 0 0.03 0.10 0.52 0.001560 NA  
## 318 BBB BB 0 0.06 0.10 0.52 0.003120 NA  
## 319 BB BB 0 0.10 0.10 0.52 0.005200 NA  
## 320 B BB 0 0.25 0.10 0.52 0.013000 NA  
## 321 C BB 0 0.01 0.10 0.52 0.000520 NA  
## 322 0 BB 0 0.52 0.10 0.52 0.027040 NA  
## 323 DD B 0 0.03 0.25 0.52 0.003900 NA  
## 324 7 B 0 0.03 0.25 0.52 0.003900 NA  
## 325 BBB B 0 0.06 0.25 0.52 0.007800 NA  
## 326 BB B 0 0.10 0.25 0.52 0.013000 NA  
## 327 B B 0 0.25 0.25 0.52 0.032500 NA  
## 328 C B 0 0.01 0.25 0.52 0.001300 NA  
## 329 0 B 0 0.52 0.25 0.52 0.067600 NA  
## 330 DD C 0 0.03 0.01 0.52 0.000156 NA  
## 331 7 C 0 0.03 0.01 0.52 0.000156 NA  
## 332 BBB C 0 0.06 0.01 0.52 0.000312 NA  
## 333 BB C 0 0.10 0.01 0.52 0.000520 NA  
## 334 B C 0 0.25 0.01 0.52 0.001300 NA  
## 335 C C 0 0.01 0.01 0.52 0.000052 NA  
## 336 0 C 0 0.52 0.01 0.52 0.002704 NA  
## 337 DD 0 0 0.03 0.52 0.52 0.008112 NA  
## 338 7 0 0 0.03 0.52 0.52 0.008112 NA  
## 339 BBB 0 0 0.06 0.52 0.52 0.016224 NA  
## 340 BB 0 0 0.10 0.52 0.52 0.027040 NA  
## 341 B 0 0 0.25 0.52 0.52 0.067600 NA  
## 342 C 0 0 0.01 0.52 0.52 0.002704 NA  
## 343 0 0 0 0.52 0.52 0.52 0.140608 NA

Ahora , para correr la funcion score en las 343 filas sería

for (i in 1:nrow(combos)) {  
 symbols <- c(combos[i, 1], combos[i, 2], combos[i, 3])  
 combos$prize[i] <- score(symbols)  
}  
head(combos, 3)

## Var1 Var2 Var3 prob1 prob2 prob3 prob prize  
## 1 DD DD DD 0.03 0.03 0.03 2.7e-05 800  
## 2 7 DD DD 0.03 0.03 0.03 2.7e-05 0  
## 3 BBB DD DD 0.06 0.03 0.03 5.4e-05 0

sum(combos$prize \* combos$prob)

## [1] 0.538014

Faltaba incluir los diamentes en la funcion score por lo que tendremos

score <- function(symbols) {  
diamonds <- sum(symbols == "DD")  
cherries <- sum(symbols == "C")  
# identify case  
# since diamonds are wild, only nondiamonds  
# matter for three of a kind and all bars  
slots <- symbols[symbols != "DD"]  
same <- length(unique(slots)) == 1  
bars <- slots %in% c("B", "BB", "BBB")  
# assign prize  
if (diamonds == 3) {  
 prize <- 100  
} else if (same) {  
 payouts <- c("7" = 80, "BBB" = 40, "BB" = 25, "B" = 10, "C" = 10, "0" = 0)  
 prize <- unname(payouts[slots[1]])  
} else if (all(bars)) {  
 prize <- 5  
} else if (cherries > 0) {  
# diamonds count as cherries  
# so long as there is one real cherry  
 prize <- c(0, 2, 5)[cherries + diamonds + 1]  
} else {  
 prize <- 0  
}  
# double for each diamond  
prize \* 2^diamonds  
}

El ciclo se actualizaría de la siguente forma

for (i in 1:nrow(combos)) {  
 symbols <- c(combos[i, 1], combos[i, 2], combos[i, 3])  
 combos$prize[i] <- score(symbols)  
}  
sum(combos$prize \* combos$prob)

## [1] 0.934356

### While loops

while (condition) {

code

}

broke <- function(start\_with) {  
 cash <- start\_with  
 n <- 0  
 while (cash > 0) {  
 cash <- cash - 1 + play()  
 n <- n + 1  
 }  
 n  
}

### Repeat loops

Se ejecutan hasta encontrar el comando break

plays\_till\_broke <- function(start\_with) {  
 cash <- start\_with  
 n <- 0  
 repeat {  
 cash <- cash - 1 + play()  
 n <- n + 1  
 if (cash <= 0) {  
 break  
 }  
 }  
 n  
}  
plays\_till\_broke(100)

## [1] 100

## Capítulo 10: Velocidad

### Vectorized code

Se pueden observar los siguientes códigos. El primero no está vectorizado. Usa un loop para manipular cada individio del vector individualmente

abs\_loop <- function(vec){  
 for (i in 1:length(vec)) {  
 if (vec[i] < 0) {  
 vec[i] <- -vec[i]  
 }  
 }  
 vec  
}

Usa logical subsetting para manipular cada elemento del vector al mismo tiempo

abs\_sets <- function(vec){  
 negs <- vec < 0  
 vec[negs] <- vec[negs] \* -1  
 vec  
}

Lo que R ejecuta más rápidamente es lo siguiente

logical tests, subsetting, and element-wise execution

Comprobemos el cambio en velocida de la siguiente forma

long <- rep(c(-1, 1), 5000000) #rep repeats a value, or vector of values, many times To use rep, give it a vector of values and then the number of times to repeat the vector. R will return the results as a new, longer vector.  
  
system.time(abs\_loop(long))

## user system elapsed   
## 0.81 0.00 0.89

system.time(abs\_sets(long))

## user system elapsed   
## 0.20 0.08 0.29

Algunas funciones ya vienen optimizadas, como el caso de la función abs, que realiza lo mismo que las dos anteriores

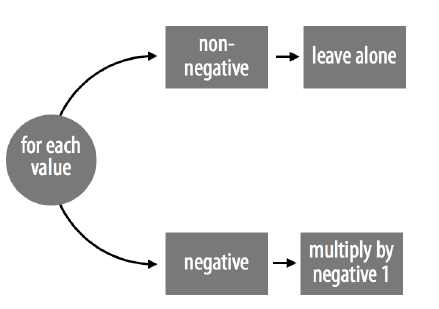
system.time(abs(long))

## user system elapsed   
## 0.02 0.04 0.05

### How to Write Vectorized Code

Vectorized code is easy to write in R because most R functions are already vectorized. Code based on these functions can easily be made vectorized and therefore fast. To create vectorized code:

1. Use vectorized functions to complete the sequential steps in your program.
2. Use logical subsetting to handle parallel cases. Try to manipulate every element in a case at once



vec <- c(1, -2, 3, -4, 5, -6, 7, -8, 9, -10)  
vec < 0

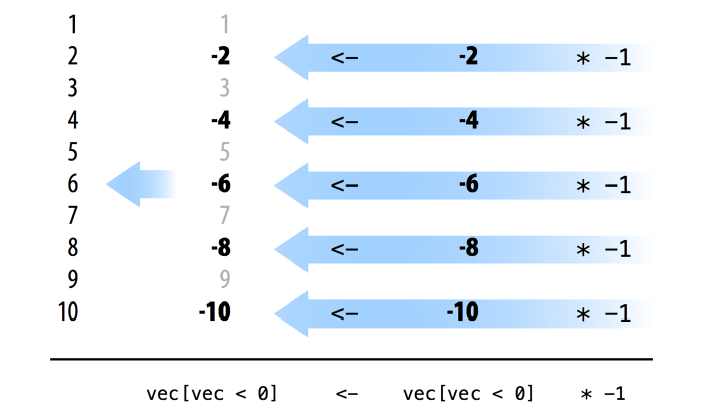
## [1] FALSE TRUE FALSE TRUE FALSE TRUE FALSE TRUE FALSE TRUE

vec[vec < 0]

## [1] -2 -4 -6 -8 -10

vec[vec < 0] \* -1

## [1] 2 4 6 8 10



#### Ejercicio:

Vectorizar el siguiente código:

change\_symbols <- function(vec){  
 for (i in 1:length(vec)){  
 if (vec[i] == "DD") {  
 vec[i] <- "joker"  
 } else if (vec[i] == "C") {  
 vec[i] <- "ace"  
 } else if (vec[i] == "7") {  
 vec[i] <- "king"  
 }else if (vec[i] == "B") {  
 vec[i] <- "queen"  
 } else if (vec[i] == "BB") {  
 vec[i] <- "jack"  
 } else if (vec[i] == "BBB") {  
 vec[i] <- "ten"  
 } else {  
 vec[i] <- "nine"  
 }  
 }  
 vec  
}  
vec <- c("DD", "C", "7", "B", "BB", "BBB", "0")  
change\_symbols(vec)

## [1] "joker" "ace" "king" "queen" "jack" "ten" "nine"

## "joker" "ace" "king" "queen" "jack" "ten" "nine"  
many <- rep(vec, 1000000)  
system.time(change\_symbols(many))

## user system elapsed   
## 9.12 0.06 9.94

## user system elapsed  
## 30.057 0.031 30.079

Vectorizado

vec[vec == "DD"]

## [1] "DD"

## "DD"  
vec[vec == "C"]

## [1] "C"

## "C"  
vec[vec == "7"]

## [1] "7"

## "vec[vec == "B"]  
## "B"  
vec[vec == "BB"]

## [1] "BB"

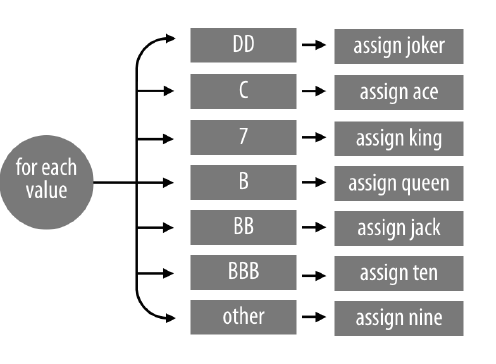
## "BB"  
vec[vec == "BBB"]

## [1] "BBB"

## "BBB"  
vec[vec == "0"]

## [1] "0"

## "0"



entonces

change\_vec <- function (vec) {  
vec[vec == "DD"] <- "joker"  
vec[vec == "C"] <- "ace"  
vec[vec == "7"] <- "king"  
vec[vec == "B"] <- "queen"  
vec[vec == "BB"] <- "jack"  
vec[vec == "BBB"] <- "ten"  
vec[vec == "0"] <- "nine"  
vec  
}  
system.time(change\_vec(many))

## user system elapsed   
## 0.47 0.07 0.53

## user system elapsed  
## 1.994 0.059 2.051

O usando una lookup table

change\_vec2 <- function(vec){  
 tb <- c("DD" = "joker", "C" = "ace", "7" = "king", "B" = "queen", "BB" = "jack", "BBB" = "ten", "0" = "nine")  
 unname(tb[vec])  
}  
system.time(change\_vec(many))

## user system elapsed   
## 0.31 0.13 0.46

## user system elapsed  
## 0.687 0.059 0.746

### How to Write Fast for Loops in R

First, do as much as you can outside of the for loop. Every line of code that you place inside of the for loop will be run many, many times. If a line of code only needs to be run once, place it outside of the loop to avoid repetition.