OPERACIONES ARITMÉTICAS

Las operaciones aritméticas son: suma, resta, multiplicación, división, potenciación, división entera.

+, -, *, /, ^, %/% or %%

OPERADORES DE COMPARACIÓN

OPERADOR	SIGNIFICADO
>	mayor que
<	menor que
>=	mayor o igual
<=	menor o igual
==	igual que
!=	distinto que

OPERADORES LÓGICOS

Los operadores lógicos son: "y", "o", "no" y "o exclusivo".

```
e1 & e2
e1 | e2
! e1
xor(e1, e2)
```

ALGUNAS FUNCIONES

```
exp(x)
log(x, base=exp(1))
log10(x)
sqrt(x)                    raíz cuadrada
abs(x)                     valor absoluto
ceiling(x)                 menor entero >= x.
floor(x)                   mayor entero<= x.
trunc(x)                  entero más cercano a x entre x y 0, inclusive,
e.g., trunc(1.5) es 1, y trunc(-1.5) es -1. trunc es como floor para
valores positivos y como ceiling para valores negativos
```

OPERADOR DE ASIGNACIÓN

Asigna un valor a un nombre.

```
<-
expresión <- valor
```

PARA GENERAR UN VECTOR

Para obtener el vector (x1,...,xn) se usa c(x1,...,xn)que concatena los elementos x1,...,xn generando un vector.
Ej: c(1,2,5) genera el vector (1,2,5)
En general c(x1,...xn) concatena los elementos x1,...xn en una lista si los xi no son todos del mismo tipo.

FUNCIÓN SAMPLE

sample(x,size, replace=FALSE,prob)

Toma una muestra de tamaño "size" de elementos de "x" con o sin repetición según se indique TRUE o FALSE.

"prob" es un vector adicional que indica la probabilidad de obtener c/u de los elementos que figuran en "x", si no se pone nada se asume que todos tienen la misma probabilidad.

Ejemplos
sample(1:k,n) genera n números al azar entre 1 y k (si n<=k)
sample(1:k,n,T) genera n números al azar con repetición entre 1 y k (si n>k)
sample(c(10,20,30,40,50),3) elige al azar una permutación de 3 números entre 10,20,30,40 y 50 sin repetición.

Si no se indica "size" ni "replace" asume que "size" es la longitud del vector x y por default asume replace =FALSE
sample(6) genera una permutación sin repetición de los 6 números 1,2 3, 4, 5 y 6

FUNCIÓN SUM

sum() devuelve el resultado de la suma de los valores presentes en el argumento
Ejemplos
sum(1:5) devuelve 15 que es el resultado de sumar los naturales de 1 a 5
sum(c(2,3,5,7)) devuelve 17 que es el resultado de la suma de los elementos que figuran en el argumento
sum(c(2,3,5,7)==c(2,4,5,6)) devuelve 2 que es el número de elementos coincidentes entre los dos vectores

USO DE FOR

for(i in 1:n) recorre los números naturales desde 1 hasta n

Ejemplo: con el siguiente algoritmo se obtiene el resultado de sumar los números naturales del 1 al 5

```
suma<-0
for(i in 1:5)
   {
    suma<-suma+i
    }
suma
```

USO DE IF

if(condición) expresión

ejemplo: if(x>0) y<-1, le asigna a y el valor 1 si x es mayor que 0

if(cond) expresión else expresión alternativa

ejemplo: if(x>0) y<-1 else y<-0, le asigna a la variable y 1 si x es mayor que 0 y el valor 0 en caso contrario

PARA OBTENER SECUENCIAS

seq: Crea un vector de números equiespaciados. El principio, el fin , el espacio entre dos números consecutivos o la cantidad de números de la secuencia pueden ser especificados

Generación de secuencias

1)from:to
2)seq(from, to)
3)seq(from, to, by=)
4)seq(from, to, length=)
5)seq(along)

Ejemplos:

```
1.1)
>1:5
[1] 1 2 3 4 5
1.2)
> 5:1
[1] 5 4 3 2 1
1.3)
> 1.1:5
[1] 1.1 2.1 3.1 4.1
2.1)
> seq(5)
[1] 1 2 3 4 5
2.2)
> seq(-5)
[1]  1  0 -1 -2 -3 -4 -5
3.1)
> seq(0, 1, 0.1)
 [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
3.2)
> seq(1,20,2)
 [1]  1  3  5  7  9 11 13 15 17 19
4.1)
> seq(-pi, pi, length=5)
[1] -3.141593 -1.570796  0.000000  1.570796  3.141593
```

rep: repite un vector x una cantidad determinada de veces (times) o hasta lograr la longitud especificada (length.out).

```
rep(x, times, length.out)
```

Ejemplos

```
rep(0,10)
 [1] 0 0 0 0 0 0 0 0 0 0
> rep(1:4,2)
[1] 1 2 3 4 1 2 3 4
```

Si times es un vector de la misma longitud de x , indica el número de repeticiones para cada componente de x .

```
>  rep(1:4,c(2,2,2,2))
[1] 1 1 2 2 3 3 4 4
> rep(1:4, length.out=18)
 [1] 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2
```

PARA HACER GRÁFICOS

plot(x,y) Si x e y son vectores de la misma longitud representa los pares ordenados con puntos

PARA ADICIONAR LÍNEAS, PUNTOS O SEGMENTOS EN UN GRÁFICO

```
lines(x, y)
points(x, y)
```

segments(x1, y1, x2, y2) adiciona un segmento a un gráfico uniendo el punto (x1,y1) con el punto (x2,y2), si x1,y1,x2,y2 son vectores de longitud k adiciona k segmentos

ARGUMENTOS OPCIONALES EN PLOT

plot(x,y,type="")
type= "p", "l", "b", "o", "s" , "h" y "n", produce puntos, lineas, ambos, ambos superpuestos, escalones , barras verticales o nada
plot(x,y,xlim=,ylim=)
xlim e ylim son vectores que dan los límites para el recorrido de x e y en el gráfico
Ej: plot(x,y,xlim=c(1,5),ylim=c(0,0.3))

PARA AGREGAR UN TÍTULO

title(main = "", sub = "", xlab = "", ylab = "", ...)
main (en la parte superior del gráfico)
sub (en la parte inferior del gráfico)
xlab (en el eje x)
ylab (en el eje y)

PARA AGREGAR TEXTO EN LOS MÁRGENES

mtext(text="", side=3, line=0)
side (puede ser 1,2,3 o 4 según si el texto va en la parte inferior, izquierda, superior o derecha)
line=0 lo pone del lado de afuera pegado al margen
line=k con k>0 lo pone afuera del margen a distancia k
line=k con k<0 lo pone por adentro del margen a distancia k

DISTRIBUCIONES

Instrucciones para la binomial

Si X~Bi(n,p)
para calcular P(X=k)
dbinom(k, size=n, prob=p)
Ej: Si x~Bi(5,0.1) para calcular P(X=3)
dbinom(3,5,0.1)
[1] 0.0081
para calcular P(X<=k)
pbinom(k, n, p) calcula la probabilidad acumulada
P(x<=3)
pbinom(3,5,0.1)
[1] 0.99954

Instrucciones para la hipergeométrica

Si X es el número de elementos obtenidos del tipo deseado
dhyper(x, m, n, k) calcula P(X=x) en un conjunto con m elementos del tipo deseado y n que no son del tipo deseado, cuando se extraen k elementos del conjunto
phyper(x, m, n, k) idem pero calcula P(X<=x)

Instrucciones para la binomial negativa

Si X~BN(r,p)
dnbinom(x, size=r, prob=p) calcula la probabilidad de obtener x fracasos para lograr r éxitos

pnbinom(x, size=r, prob=p) idem para calcular probabilidad de a lo sumo x fracasos

Instrucciones para la geométrica
Si X~G(p)
dgeom(x, prob=p) calcula P(X=x)
pgeom(q, prob=p) calcula P(X<=x)

Instrucciones para la distribución Poisson
Si X~P(λ)
dpois(x, lambda=λ) calcula P(X=x)
ppois(x, lambda=λ) calcula P(X<=x)

PARA ESCRIBIR NUEVAS FUNCIONES EN R

El lenguaje R permite al usuario definir objetos que sean funciones. Estas se convierten en auténticas funciones de R, que se almacenan en una forma interna y se pueden utilizar en expresiones futuras. En el proceso, el lenguaje se enriquece enormemente, ganando conveniencia y elegancia. De hecho, muchas de las funciones que vienen con el paquete R (como por ejemplo, mean(), var(), etc.) son ellas mismas funciones escritas en R. Una función se define por una asignación de la forma

> **nombre <- function(arg_1, arg_2, ...) expresion**

La "expresion" es una expresión en R (si ocupa más de un renglón puede escribirse en varios si está encerrada entre llaves {}, que utiliza los argumentos: arg_i, para calcular su valor. El valor de la expresión es devuelto como el valor de la función.

Veámoslo a través de un ejemplo. Queremos definir la función cuadrática
y = f(x) = 3x^2-5x+2.

La llamaremos función ejemplo. Como nombre de la función podemos usar cualquier palabra (que no sea una palabra ya cargada en el R, como log o sum) que puede incluir letras y puntos. Llamémossla **ejemplo.cuadratica**

```
ejemplo.cuadratica<-function(xx) {3*xx*xx-5*xx+2}
```

Luego, cuando queremos calcular f(4) simplemente escribimos en la línea de comando:

```
> ejemplo.cuadratica(4)
[1] 30
```

Veamos un ejemplo más complicado. Supongamos que queremos escribir una función que calcule las coordenadas polares de un punto cuya coordenada y es no nula.

```
coord.polar<-function(x,y)
{
r<-sqrt(x*x+y*y)
tita<-atan(x/y)
respuesta<-c(r,tita)
respuesta
```

```
}
```

Hallemos las coordenadas polares del (3,2):

```
> coord.polar(3,2)
[1] 3.6055513 0.9827937
```

Notar que la función devuelve como valor a la última expresión escrita antes de la última llave, en este caso, "respuesta".

Como último ejemplo, si queremos resolver el ejercicio 1 de la práctica 1 de laboratorio utilizando una función podemos definir

```
simulacion.moneda<-function(totales)
{
favorables<-0
for (i in 1:totales)
{
moneda<-sample(c(0,1),1,T)
favorables<-favorables+moneda
}
frecuencia<-favorables/totales
frecuencia
}
```

Cada vez que corramos llamemos a la función nos dará como resultado la proporción de caras que obtuvimos al tirar una cierta cantidad de veces la moneda. Por ejemplo

```
> simulacion.moneda(10)
[1] 0.6
> simulacion.moneda(10)
[1] 0.4
> simulacion.moneda(10)
[1] 0.7
```

Notar que las asignaciones normales realizadas dentro de las instrucciones que definen a una funciónson locales y temporarias y se pierden una vez que se sale de la función. En este caso si le pedimos al programa R que nos devuelva el valor de favorables obtenemos

```
> favorables
Error: object "favorables" not found
```

Es decir, la asignación

```
favorables<-favorables+moneda
o favorables<-0
```

no afecta el valor de "favorables" en el siguiente llamado de la función.