```
# AMPLIACIÓN DE INVESTIGACIÓN OPERATIVA.

# I.T. en INFORMÁTICA DE SISTEMA Y GESTIÓN

# PRÁCTICA: INTRODUCCIÓN AL PROGRAMA R.


# EJEMPLO: Lanzamiento de una moneda perfecta

        # 1: Cara

        # 0: Cruz

        rbinom(1,1,0.5)


# PREGUNTA: ¿Cuántos lanzamientos son requeridos para obtener 3 caras?


#  Tipos de datos.

# - Valores numéricos:

        moneda<-rbinom(1,1,0.5)

        moneda

# - Vectores: utilizaremos la función  c().

        moneda1<-rbinom(1,1,0.5); moneda1

        moneda2<-rbinom(1,1,0.5); moneda2

        moneda<-c(moneda1,moneda2); moneda

        ?length

        length(moneda)

# - Concadenación con vectores

        moneda3<-rbinom(1,1,0.5); moneda3

        moneda<-c(moneda,moneda3); moneda

# Listamos los elementos guardados

     ls()

# Borramos un elemento almacenado

     rm(moneda)

# Crear un vector físicos sin elementos.


        moneda<-c(moneda,rbinom(1,1,0.5))


        moneda<-numeric()

        moneda

        moneda<-c(moneda,rbinom(1,1,0.5))

        moneda
```



```
# - Matrices.

  # A partir de vectores columnas con la función cbind()

        grupo1<-rbinom(3,1,0.5); grupo1 # Moneda perfecta
        grupo2<-rbinom(3,1,0.75); grupo2 # Moneda favorece Cara
        grupo3<-rbinom(3,1,0.25); grupo3 # Moneda favorece Cruz

        A<-cbind(grupo1,grupo2,grupo3);A

  # A partir de vectores filas con la función rbind()

        B<-rbind(grupo1,grupo2,grupo3);B


  # A partir de la función matrix(vector,nº filas,nº columnas)

        matrix(0,2,3)

  # Se pueden concadenar matrices

        rbind(A, B)

        cbind(A, B)

  # Selección de datos:

     A
     A[1,3]
     A[1,]      # Fila primera
     A[,2]      # Columna segunda
     A[1,3]<-1  # Rectificar datos
     A


# Arrays: Matrices de matrices

        Tenemos dos marices A y B.

        AB<-array(0,c(3,3,2)); AB

        AB[,,1]<-A
        AB[,,2]<-B
        AB


# -Listas.

        grupo4<-rbinom(4,1,0.1); grupo4 # Moneda favorece Cruz


        monedas<-list("0.5"=grupo1,"0.75"=grupo2,"0.25"=grupo3,"0.1"=grupo4)

        monedas$"0.1" # Muestra los lazamientos del grupo 4 (moneda de 0.9)

        monedas[[4]] # Muestra los lazamientos del grupo 4 (moneda de 0.9)

        Concadenación de lista

        c(monedas,list("0.6"=rbinom(2,1,0.6)))


# Algunas herramientas del R.


# - Función seq(valor inicial, valor final, by=paso ó length=longitud)

     seq(1,50,1)
```

```
     seq(1,50,length=50)

# - Vectores lógicos:
#     Vectores con dos posibles elementos, T (verdadero) y  F (falso).
#     Se generan mediante condiciones.

     moneda<-rbinom(10,1,0.5)

#   a) Operadores lógicos:  <, >, <=, >=, ==, !=

    moneda==1


#   b) Si C1 y C2 son operaciones lógicas, entonces  C1 & C2 es la intersección,
#      C1 | C2 la unión y !C1 la negación.

    (moneda>=0)&(moneda<1)

    (moneda>=1)|(moneda<=0)

    moneda==1

    !(moneda==1)

#   c) Operaciones aritméticas: T se considera 1 y F se considera 0.

    sum(moneda==1)


# - Función length(vector)

     length(moneda)


# - Selección y modificación de un conjunto de datos: vectores índices.


#   -- Vector lógico: vector de la misma longitud que el vector del
#     que se seleccionan elementos. Se seleccionan los correspondientes
#     a T y no los correspondientes a F.

        1:10 # índice de los lanzamientos

        (1:10)[moneda==1] # Lanzamientos con Cara

#  -- Vector de enteros positivos: Se seleccionan los elementos
#     correspondientes a las posiciones que indica el vector.

    moneda[6:10] # Selecciona los 5 últimos lanzamiento


# - Valores perdidos (not available): NA

        0/0


# Algunas herramientas estadísticas del R.


   mean(moneda) # Nos proporciona la media muestral del vector moneda.

   median(moneda) # Nos proporciona la mediana muestral del vector moneda.

   var(moneda) # Nos proporciona la varianza muestral del vector moneda.

   sd(moneda) # Nos proporciona la desviación típica muestral del vector moneda.
```



```
   min(moneda) # Nos proporciona el valor mínimo del vector moneda.

   max(moneda) # Nos proporciona el valor máximo del vector moneda.


# Generar datos en R.

        # Variable discreta: Binomial(n,p)

        #    rbinom(números,n,p)

        rbinom(10,1,0.5) # 10 Lanzamientos de una moneda perfecta

        rbinom(10,5,0.5) # 10 experimentos donde en cada uno de ello
                         # se lanza 5 veces una moneda perfecta y se
                         # anota el número de caras.


        # Variable continua: Normal(media,sd)

        #    rnorm(números,media=0,sd=1)


        rnorm(100,7,1) # Nota de selectividad


# - Histograma de los datos simulados

     moneda<-rbinom(100,1,0.5)

     hist(moneda,main="Histograma")


     selectividad<-rnorm(100,7,1); selectividad

     hist(selectividad,main="Histograma")


# - Representar gráficamente una secuencia de valores

    indice<-1:10

    valores<-rnorm(10,7,7)


    plot(indice,valores,type="l",xlab="",ylab="",col=2)


# - Gráfico de la función de densidad de una distribución normal

     x<-seq(-3,3,0.01)

     plot(x,dnorm(x),type="l",col=2,
          main="Densidad N[0,1]", xlab="x", ylab="y")

     abline(v=0)


# Otras herramientas del R.

# Se pueden implementar nuevas funciones:

#  nombre función<-function(argumentos){sentencias ejecutables (separadas por
;)}
```

```
media<-function(x){sum(x)/length(x)}

media(1:20)


# Órdenes de control, útiles para la programación:


# Ejecución condicional:


        # if(condicion){ejecucion si true}else{ejecución si false}


        moneda<-rbinom(1,1,0.5); moneda # Lanzamiento de una moneda perfecta

        # El siguiente lanzamiento, va a estar condicionado por el previo:
        # Si es cara, entonces se lanza con la moneda 0.1 (Favorece a las
cruces)
        # Si es cruz, entonces se lanza con la moneda 0.9 (Favorece a las caras)

        moneda<-rbinom(1,1,.5);
        moneda
        if(moneda[length(moneda)]==1){moneda<-c(moneda,rbinom(1,1,0.1)); moneda}
else{moneda<-c(moneda,rbinom(1,1,0.9)); moneda}

        # Otra forma de obernerlo es mediante la sentencia

        moneda<-rbinom(1,1,.5);
        moneda

moneda<-c(moneda,(moneda[length(moneda)]==1)*rbinom(1,1,0.1)+(moneda[length(mone
da)]==0)*rbinom(1,1,0.9));moneda


# Ciclos:


# for(contador) {expresiones ejecutables}

# 10 lanzamientos del experimento anterior

        moneda<-rbinom(1,1,.5)
        moneda
        for(i in 1:10){moneda<-c(moneda,
(moneda[length(moneda)]==1)*rbinom(1,1,0.1)+
(moneda[length(moneda)]==0)*rbinom(1,1,0.9))}; moneda


# while(condición) {expresiones ejecutables}

# Determinar el número de lanzamiento que hay que realizar de una
# moneda perfecta hasta obtener "m" caras.

        moneda<-function(m,p=0.5){i<-0;cont<-0;
                while(i<m){i<-i+rbinom(1,1,p);cont<-cont+1}
                ;cont}

      moneda(10)


# Repertir el proceso anterior "n" veces
```



```
rmoneda<-function(n=1,m=1,p=0.5){res<-numeric();
                for(i in 1:n){res<-c(res,moneda(m,p))}
                ;res}

simoneda<-rmoneda(1000,10)


# Una función más eficiente que la sentencia "for".

# apply(matriz,fila o columna, función)


# La función rmoneda con apply


      rmoneda2<-function(n=1,m=1,p=0.5){apply(cbind(rep(m,n)),1,moneda,p=p)}

      rmoneda2(10,3)

      system.time(sim1<-rmoneda(10000,3))[3] # Con el for

      system.time(sim1<-rmoneda2(10000,3))[3] # Con el apply


# Para ampliar el espacio de almacenamiento del R


        ?memory.limit


# Para depurar los fallos de las funciones

        #debug()

        # Función sin errores sintácticos pero si internos


        rmoneda<-function(n=1,m=1,p=0.5){
                for(i in 1:n){res<-c(res,moneda(m,p))}
                ;res}


        debug(rmoneda)

        rmoneda(10,3)

        # Función sin errores

        rmoneda<-function(n=1,m=1,p=0.5){res<-numeric()
                for(i in 1:n){res<-c(res,moneda(m,p))}
                ;res}


        debug(rmoneda)

        rmoneda(5,3) # Ir tecleando el nombre de las variables.


# EJERCICIOS

# Determinar el comportamiento de los siguientes experimentos:


# 1. Números de lanzamientos necesarios de una moneda hasta obtener 3 caras o 3
cruces
```

```
# 2. Números de lanzamientos necesarios de una moneda hasta obtener 3 caras y 3
cruces

# 3. Números de lanzamientos necesarios de una moneda hasta obtener 3 caras
consecutivas

# 4. Números de lanzamientos necesarios de una moneda hasta obtener 3 caras o 3
cruces consecutivas
```