PRELIMINARES

NumPy es una extensión de Python, que le agrega mayor soporte para vectores y matrices, constituyendo una biblioteca de funciones matemáticas de alto nivel para operar con esos vectores o matrices. Para usarla debes importarla en tu código usando:

```
import numpy
```

En muchos códigos verán que al importar numpy se usa la siguiente línea:

```
import numpy as np
```

Esto permita usar np en vez de numpy al llamar a una función de la biblioteca. Usaremos esta notación por comodidad.

La "gracia" de NumPy es poder ocupar arreglos en vez de listas. Los arreglos son contenedores de información, al igual que las listas, pero mantiene solamente un tipo de datos: o todos los valores son enteros, o todos son decimales, o todos textos, etc.

Para crear un arreglo en NumPy hay varias opciones:

- np.array([lista de elementos]): Crea un arreglo con los elementos de la lista
- np.empty([numElem]): Crea un arreglo con numElem elementos que no tienen valor inicial
- np.zeros([numElem]): Crea un arreglo con numElem elementos y le asigna a cada elemento el valor 0
- np.ones([numElem]): Crea un arreglo con numElem elementos y le asigna a cada elemento el valor 1
- np.full([numElem], valor): Crea un arreglo con numElem elementos y le asigna a cada elemento el valor valor
- np.random.random([numElem]): Crea un arreglo con numElem elementos y le asigna a cada elemento un valor random entre 0 y 1
- np.random.randint(inicio, fin, [numElem]): Crea un arreglo con numElem elementos y le asigna a cada elemento un valor entero random entre inicio y fin

Para acceder a un elemento de estos arreglos se utiliza la misma nomenclatura que en las listas nombreArreglo[indice]. Al acceder a este elemento es posible obtener su valor o modificarlo.
Ejemplo:

```
import numpy as np

arreglo1 = np.array([1,2,3,4,5])
arreglo1[2] = 10
print(arreglo1)    #muestra por pantalla [1 2 10 4 5]
print(arreglo[3])  #muestra por pantalla 4
```

El ser todos los datos del mismo tipo nos permite aplicar funciones en forma directa sobre un arreglo. NumPy provee de varias funciones de utilidad para ello:

NumPy da acceso a múltiples funciones matemáticas, entre las que se encuentran:

- np.sum(x): retorna la suma de todos los elementos del arreglo x
- np.prod(x): retorna la multiplicación de todos los elementos del arreglo x
- np.mean(x): retorna el promedio de los elementos del arreglo x
- np.min(x): retorna el mínimo de los elementos de x
- np.max(x): retorna el máximo de los elementos de x
- np.argmin(x): retorna la posición del elemento mínimo de x
- np.argmax(x): retorna la posición del elemento máximo de x
- np.abs(x): retorna un arreglo con los valores absolutos de los elementos de x
- np.round(x, numero_decimales): retorna un arreglo que contiene los elementos de x redondeados al número de decimales dado
- np.truncate(x): retorna un arreglo que contiene la parte entera de los elementos de x
- np.square(x): retorna un arreglo que contiene los cuadrados de los elementos de x
- np.sqrt(x): retorna un arreglo que contiene las raíces cuadradas de los elementos de x

Además NumPy permite hacer cálculos entre arreglos SIEMPRE Y CUANDO tengan la misma cantidad de elementos. Los arreglos se pueden multiplicar, sumar, restar, dividir, comparar, etc.

Ejemplo:

```
import numpy as np

arreglo1 = np.array([1,2,3,4,5])
arreglo2 = np.zeros(5)

suma = arreglo1 + arreglo2
print(suma)

compara = arreglo1 > arreglo2
print(compara)
```

Las comparaciones entregan un valor de verdad (Verdadero o Falso) para cada elemento, dependiendo de la condición.
Si queremos usar más de una condición podemos usar numpy.logical_and(lista, de, condiciones):

```
import numpy as np

arreglo1 = np.array([1,-2,3,-4,5])
arreglo2 = np.zeros(5)

compara = np.logical_and(arreglo1 > arreglo2, arreglo1 % 2 == 0)
print(compara)
```

Esto es muy útil pues ahora podemos crear un arreglo usando las condiciones como un filtro, pásandolo como posiciones de un arreglo. Por ejemplo, si queremos los valores del arreglo1 que sean mayores al arreglo2 y sean impares usamos:

```
import numpy as np

arreglo1 = np.array([1,-2,3,-4,5])
arreglo2 = np.zeros(5)

compara = np.logical_and(arreglo1 > arreglo2, arreglo1 % 2 != 0)
print(arreglo1[compara])
```

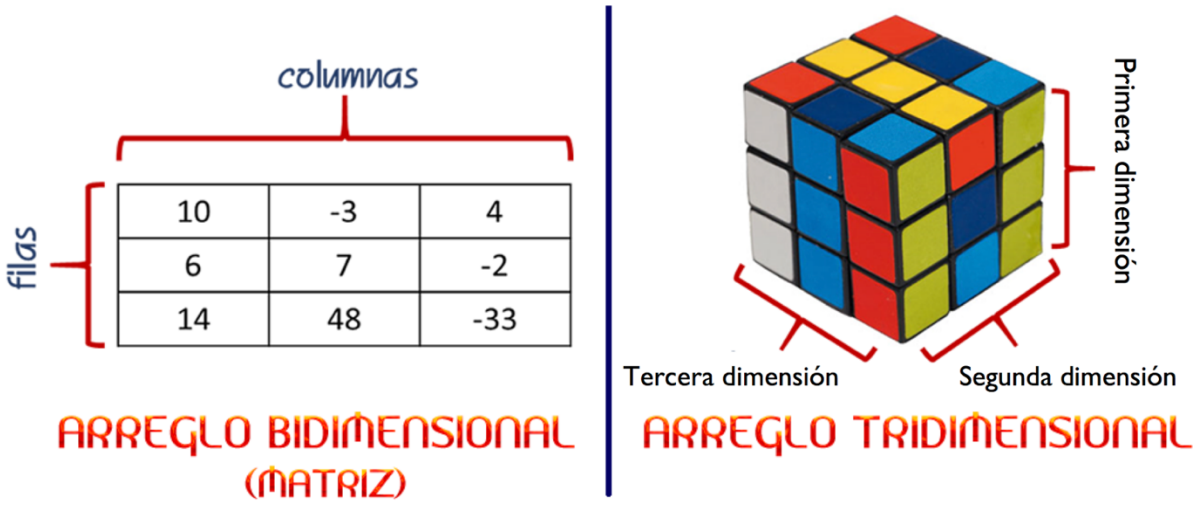
Por último, nos permite ver si todos o alguno de los elementos cumple la condición usando np.all() o np.any() respectivamente:

```
import numpy as np

arreglo1 = np.array([1,-2,3,-4,5])
arreglo2 = np.zeros(5)

compara =arreglo1 > arreglo2
print(np.all(compara)) # False, no todos los elementos de arreglo1 son mayores al elemento en arreglo2
print(np.any(compara)) # True, al menos un elementos de arreglo1 es mayor al elemento en arreglo2
```

Aunque los arreglos puedan ayudarnos a resolver varios problemas, el que sean de una sola dimensión reduce su aplicabilidad. Para resolver este problema existen arreglos multidimensionales.



Una matriz (arreglo bidimensional) tiene un número determinado de filas y columnas, los que son determinados en el momento de su creación.

Su creación y manejo es similar a la creación unidimensional, pero con dos índices. Por ejemplo, para crear un arreglo bidimensional con valores zeros en NumPy:

- np.zeros([numElem1,numElem2]): Crea un arreglo con numElem1 filas y numElem2 columnas y le asigna a cada elemento el valor 0

En forma similar, todas las otras funciones existentes para crear arreglos se pueden utilizar. Para ello, basta con agregar el número de dimensiones el número de elementos que se requiere en cada dimensión.

En forma similar, para acceder al dato i,j de la matriz llamada MATRIZ, escribimos MATRIZ[i,j], donde i hace referencia a la fila de la matriz y j hace referencia a la columna. Tal como en el caso de una dimensión, al acceder a este elemento es posible obtener su valor o modificarlo.
Ejemplo:

```
import numpy as np

arreglo1 = np.zeros([2,2])
arreglo1[0,0] = 10
print(arreglo1)     #muestra por pantalla [[10 0]
                                          [ 0 0]]
print(arreglo[0,1])  #muestra por pantalla 0
```

Para obtener el número de elementos de la matriz podemos utilizar el atributo size. Sin embargo, el número de elementos por dimensión de la matriz está dado por el atributo shape, el que retorna un "arreglo" con el número de filas y columnas.

El operador ":", permite obtener acceso a multiples elementos de la matriz, e incluside un arreglo unidimensional. En forma particular el operador arreglo[i1:i2] retorna un arreglo con los elementos correspondiente a los índices i1 a i2-1. En el caso de una matriz, el operador matriz[i1:i2 , j1:j2] retorna la submatriz correspondiente a todos los elementos entre las filas i1 a i2-1 y las columnas j1:j2-1.

Además, el operador [i,:] permite obtener/modificar todos los elementos de la i-ésima fila, mientras que el operador [:,i] permite obtener/modificar todos los elementos de la i-ésima columna.

Ejemplo, calcular el promedio simple de 40 alumnos donde cada alumno tienen 3 notas generadas en forma aleatoria

```
import numpy
notas = numpy.random.random([40,3])*6+1      #matriz con notas aleatorias
prom = numpy. zeros([40])                    #arreglo para guardar los promedios
for i in range(0,notas.shape[0]):
     prom[i] = notas[i,:].mean()             #promedios para cada alumno
print("Los promedios son ", prom.round(2))
```

Como se puede observar, al ser todos los datos del mismo tipo, numpy nos permite aplicar las mismas funciones mencionadas anteriormente tanto en matrices como en cierta columna o fila. Por ejemplo:

- **matriz*num**: retorna una matriz donde todos los elementos son multiplicados por num (pueden usar /, +, -)
  matriz[i,:]*num: retorna un arreglo correspondiente a la i-ésima fila multiplicada por num
  matriz[:,i]*num: retorna un arreglo correspondiente a la i-ésima columna multiplicada por num

- **matriz.min():** retorna el valor mínimo de la matriz
  matriz[i,:].min(): retorna el valor mínimo de la i-ésima fila
  matriz[:,i].min(): retorna el valor mínimo de la i-ésima columna

El resto de las funciones también pueden ser aplicadas de esta misma manera.

Finalmente, a diferencia de las variables simples, donde la instrucción A = B crea una variable nueva A con el valor de la variable B. Tanto listas como arreglos unen las dos variables a la misma lista o arreglo. Por lo tanto, para generar una copia exacta del mismo arreglo es necesario utilizar la función **numpy.copy(A)**, la cual replica el arreglo A y retorna un nuevo arreglo con esta información.

PROBLEMAS PROPUESTOS

1. Crea un arreglo de 100 números aleatorios y luego modifica el arreglo con los valores desplazados, de tal forma que el mayor sea 1. Es decir, si el arreglo de números aleatorios tiene como valor máximo el 0.7, entonces debes sumarle a cada elemento 0.3. Luego imprime el arreglo en pantalla.

   Una solución posible:

```
import numpy as np

numeros = np.random.random([100])
maximo = np.max(numeros)
numeros = numeros + (1-maximo)
print(numeros)
```

2. Crea un arreglo de 100 números aleatorios y modifica el arreglo de tal forma que el promedio esté en cero. Luego cuenta cuantos valores hay sobre cero y cuantos bajo cero, e imprime eso en pantalla.

   Una solución posible:

```
import numpy as np

numeros = np.random.random([100])
promedio = np.mean(numeros)
numeros = numeros - promedio
print('Bajo cero', np.sum(numeros < 0))
print('Sobre cero', np.sum(numeros > 0))
```

3. Crea un arreglo de 100 números aleatorios y suma solamente aquellos valores que sean mayores a 0.5.

   Una solución posible:

```
import numpy as np

numeros = np.random.random([100])
print('La suma es', np.sum(numeros[numeros > 0.5]))
```

4. Crea un arreglo de 100 números enteros aleatorios entre 1 y 10000 y encuentra el más cercano a 6543. Imprime el valor en pantalla.

   Una solución posible:

```
import numpy as np

numeros  = np.random.randint(1, 10000, [100])
promedio = np.mean(numeros)
copia    = np.abs(numeros - 6543)
posicion = np.argmin(copia)
print('El número más cercano a 6543 es', numeros[posicion])
```

5. Crea un arreglo de tamaño 100 con número aleatorios, retorna el valor máximo y luego reemplaza el valor en la posición del máximo en el arreglo por un cero.

Una solución posible:

```
import numpy as np

numeros = np.random.random([100])
print(numeros)
print('El elementos más grande es', np.max(numeros))
numeros[np.argmax(numeros)] = 0
print(numeros)
```

6. Escribe el código en Python que crea un arreglo 1D de 10 elementos con valores aleatorios enteros entre 0 y 9 y que despliegue el contenido del arreglo, así como el número de valles y crestas contenidos en el. Se definen como valles los números que son menores que sus dos vecinos inmediatos en el arreglo (a izquierda y derecha). Se definen como crestas los números que son mayores que sus dos vecinos inmediatos (a izquierda y derecha) en el arreglo. Por ejemplo:

```
[7.4  0.8  6.8  8.3  3.0  5.1  6.3  4.4  8.3  6.2]
Este arreglo tiene  3 valles y  3 crestas
```

Una solución posible:

```
import numpy

#crea arreglo con 50 enteros aleatorios y redondea a 1 decimal
N=numpy.random.random([10])*9
N=N.round(1)
print(N)

#recorre arreglo y detecta valles y crestas
valles=0
crestas=0
for i in range(1,N.size-1):
    if(N[i]<N[i-1] and N[i]<N[i+1]):
        valles=valles+1
    if(N[i]>N[i-1] and N[i]>N[i+1]):
        crestas=crestas+1
print("Este arreglo tiene ",valles,"valles y ", crestas, "crestas")
```

7. Escribe el código Python que calcula el producto punto entre dos vectores a y b de largo 5. Los vectores se crean aleatoriamente. El producto punto se define como:

$$a \cdot b = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$$

Por ejemplo:

```
[2  9  3  10 10]
[3  7  5  1  6]
El producto punto es igual a  154.0
```

Una solución posible:

```
import numpy

#crea vectores y los despliega
a=numpy.random.random([5])*10
b=numpy.random.random([5])*10
a=a.round(0)
b=b.round(0)
print(a)
print(b)

#calcula producto punto
pdto_pto=0
for i in range(0, a.size):
    pdto_pto=pdto_pto+a[i]*b[i]

print("El producto punto es igual a ", pdto_pto)
```

8. Las agencias de pluviometría de las ciudades de Santiago y Mendoza sospechan que el cambio climático podría estar alterando el patrón de precipitaciones de estas ciudades, haciéndolas más parecidas en términos de ocurrencias de lluvias. Para verificar esta sospecha ambas agencias registraron la existencia de lluvias para cada día del año 2017.

   Las agencias necesitan que construyas un programa en Python que despliegue por pantalla el número de días del año 2017 en que **en ambas ciudades** se presentó el mismo patrón de lluvias (es decir, en ambas llovió el mismo día o en ambas no llovió el mismo día). Para esto, el programa debe manejar **dos arreglos unidimensionales** (llamados Santiago y Mendoza) de 365 casillas cada uno. La casilla 0 representa el 1 de Enero, la casilla 1 el 2 de Enero y así sucesivamente. En cada casilla se almacena un 1 si hubo lluvias y 0 si no las hubo.

   Escribe un programa que despliegue por pantalla el número de días en que llovió en ambas ciudades simultáneamente y el número de días en que no llovió en ambas ciudades simultáneamente ciudades. Emula el contenido de los arreglos generando el contenido de manera aleatoria.

Una solución posible:

```
import numpy

#crea arreglos Santiago y mendoza con registros (1: lluvia, 0: no lluvia)
Santiago = numpy.random.random([365])
Mendoza  = numpy.random.random([365])

Santiago = Santiago.round(0)
Mendoza  = Mendoza.round(0)

print(Santiago)
print(Mendoza)

#contador de dias en que llovio en ambas ciudades
lluvia=0

#contador de dias en que no llovio en ambas ciudades
no_lluvia=0

#recorre arreglos y cuenta los dias con igual patron
for i in range(0,Santiago.size):
      if(Santiago[i]==Mendoza[i] and Santiago[i]==0):
              no_lluvia=no_lluvia+1
      if(Santiago[i]==Mendoza[i] and Santiago[i]==1):
              lluvia=lluvia+1

print("LLovio en ambas ciudades: ",lluvia,"dias")
print("No lLovio en ambas ciudades:",no_lluvia,"dias")
```

9. En un arreglo de 10 posiciones se simula una competencia entre dos bandos. Las posiciones 0-4 son las del bando A y las posiciones 5-9 son las del bando B. La competencia se realiza de la siguiente manera:
   - En la primera ronda se enfrentan las posiciones más cercanas (posición 4 del bando A y posición 5 del bando B). Gana el bando con el número mayor.
   - Luego se enfrentan las siguientes posiciones más cercanas (posición 3 del bando A y posición 6 del bando B) y así sucesivamente hasta el último enfrentamiento entre las posiciones 0 y 9.

   Escribe un programa en Python que genere el arreglo de 10 posiciones con números aleatorios entre 0 y 9 y luego vaya mostrando por pantalla cómo se va desarrollando la competencia y qué bando la gana. Por ejemplo:

```
[3. 1. 3. 3. 4. 7. 4. 3. 5. 2.]
Enfrentamiento  1
Gana bandoB
Enfrentamiento  2
Gana bandoB
Enfrentamiento  3
Empate
Enfrentamiento  4
Gana bandoB
Enfrentamiento  5
Gana bando A
======================
Puntaje bando A:  1
Puntaje bando B:  3
Bando ganador:
B
```

Una solución posible

```
import numpy

#crea arreglo para competencia
a=numpy.random.random([10])*9
a=a.round(0)
print(a)

#registro de puntos
bandoA=0
bandoB=0

print(int(a.size/2)-1)

#ciclo que controla los enfrentamientos
for i in range(int(a.size/2),a.size):
    print("Enfrentamiento ",int(i-a.size/2+1))
    if(a[i]<a[a.size-i-1]):
       bandoA=bandoA+1
       print("Gana bando A")
    else:
        if(a[i]>a[a.size-i-1]):
            bandoB=bandoB+1
            print("Gana bandoB")
        else:
            print("Empate")

print("======================")
print("Puntaje bando A: ", bandoA)
print("Puntaje bando B: ", bandoB)

print("Bando ganador: ")

if(bandoA>bandoB):
        print("A")
else:
    if(bandoB>bandoA):
        print("B")
    else:
        print("Empate")
```

10. Escribe el código en Python que despliega en pantalla el número de veces que aparece cada número contenido en un arreglo 1D de 20 elementos enteros. El arreglo se llena con números enteros aleatorios entre 0 y 5. Por ejemplo:

```
[2. 1. 2. 1. 3. 2. 2. 5. 3. 2. 3. 2. 3. 0. 1. 2. 3. 2. 3. 3.]
Frecuencia del numero  2.0 :  8
Frecuencia del numero  1.0 :  3
Frecuencia del numero  3.0 :  7
Frecuencia del numero  5.0 :  1
Frecuencia del numero  0.0 :  1
```

Una solución posible

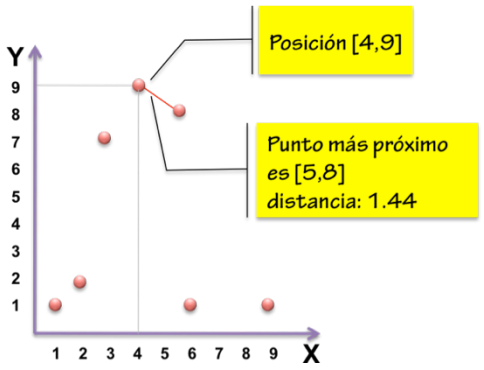
```
import numpy

#crea arreglo
a=numpy.random.random([20])*5
a=a.round(0)
print(a)

#recorre arreglo, cada vez compara con el resto de los numeros
for i in range(0, a.size-1):
    iguales=0
    #verifica que el numero a[i] no haya sido revisado antes
    revisar=1
    for k in range(0,i):
        if(a[i]==a[k]):
                revisar=0
    #calcula frecuencia solo si el numero es "nuevo"
    if(revisar==1):
        for j in range(i,a.size):
            if(a[i]==a[j] and revisar==1):
                iguales=iguales+1
        print("Frecuencia del numero ",a[i],": ",iguales)
```

19. Suponga que tiene dos vectores de igual tamaño de largo 10 que representan puntos en el espacio cartesiano X-Y. Realice un programa que por cada coordenada determine la coordenada más cercana. Indique dicha distancia. (✋: *utilice ciclos dobles*). Ocupe los siguientes datos de entrada:

```
X=numpy.array([2, 3, 6, 6, 4, 1, 9])
Y=numpy.array([1, 7, 1, 8, 9, 1, 1])
```



20. La gran multitienda LeroMart tiene una oferta lleve 3 y pague 2 la cual consiste en poder comprar 3 elementos cuales quiera y solo pagar 2 (la tienda no es tonta por lo que cobrará los 2 elementos más caros), la promoción no aplica si lleva más de 3 elementos. Raspum tiene una lista de N

elementos que desea comprar (asuma que todos los elementos existen en LeroMart). Raspum es muy inteligente por lo que decide entrar las veces que sean necesarias para pagar menos. Por ejemplo si entra 2 veces comprando 3 elementos cada vez, pagará sólo 4 elementos en lugar de pagar los 6.

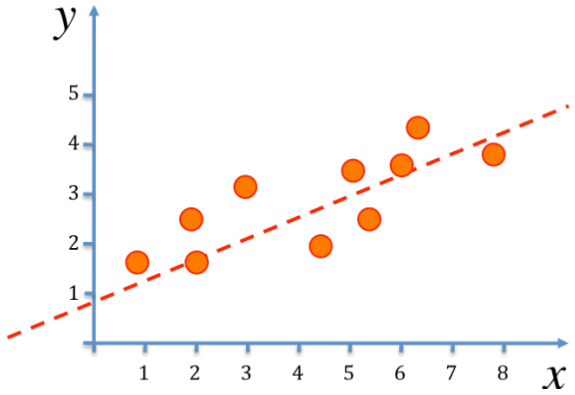
Ejemplo:

```
cuantos productos? 7
2000 4000 1000 2000 4000 3000 2000

El valor mínimo que podría pagar es: 13000
```

Realice un programa en C que solicite el número de productos a comprar y luego solicite precios de cada uno (el largo no puede ser superior a 20 productos). Finalmente calcule el valor mínimo que podría pagar Raspum empleando la estrategia descrita anteriormente.

21. Usted dispone de un conjunto de datos en una matriz de 10 x 2, donde la primera columna corresponde a la posición del eje x y la segunda columna es la posición del eje y. Realice un programa que ajuste dichos datos a una ecuación de la recta, es decir, encuentre los parámetros m, b de la ecuación de la recta $y = m \cdot x + b$.



	x	y
1	0.9	1.7
2	2.1	1.5
3	1.9	2.5
4	3	3.1
5	4.5	1.8
6	5	3.5
7	5.4	2.2
8	6	3.3
9	6.2	4.5
10	7.9	3.9

Para resolver este problema, de las estadísticas sabemos que m y b se resuelven de la siguiente forma:

$$m = \frac{n \cdot S_{XY} - S_X S_Y}{n \cdot S_{XX} - S_X S_X}$$
$$b = \frac{S_{XX} S_Y - S_X S_{XY}}{n S_{XX} - S_X S_X}$$

donde:

$$S_X = \sum_{i=1}^{n} x_i \quad S_Y = \sum_{i=1}^{n} y_i \quad S_{XY} = \sum_{i=1}^{n} x_i \cdot y_i \quad S_{XX} = \sum_{i=1}^{n} x_i^2$$

El símbolo Σ significa sumatoria, es decir, la suma de valores parciales, y n es el número de datos del

22. El invierno se acerca rápidamente y usted muy precavido ha decidido instalar un nuevo sistema de calefacción en su hogar. Para ello usted cotiza distintos modelos de calefacción, desde estufas a parafina, sistemas eléctricos, calefacción a leña, termopanel, etc. (Tabla 1)

Tabla 1. Cotizaciones de calefacción de 10 sistemas

Sistema	Costo mensual (c)	Calor (B)	Área (A)
1	$51.000	5000	35
2	$37.500	3500	25
3	$42.700	3400	30
4	$67.000	5100	38
5	$15.000	2100	15
6	$35.000	3600	28
7	$28.000	3250	25
8	$17.000	2300	12
9	$43.000	3700	32
10	$29.000	3100	30

Cada uno de estos dispositivos tiene un costo de mantención diario (d) , un rendimiento de calor por hora (B) y un rendimiento por área cuadrada (A). Dado sus conocimientos en programación, diseñe un programa que busque el mejor sistema tal que minimice la ecuación de rendimiento y maximice el área cubierta por el sistema.

$$\text{rendimiento} = \frac{d \cdot A}{B},$$

donde d es el costo diario, c es el costo mensual ($c = d \cdot 30$). Su programa debe emplear los datos de la tabla 1 almacenados en una matriz. Al finalizar, su programa debe indicar cuál es el mejor sistema.

23. Este es el juego del gato y el ratón. Usted dispone de una malla cuadrada de 5×5 y comienza en la posición (1,1). El ratón se encuentra en una posición aleatoria de la malla (definida con la función `random.randint`) . A medida que usted se mueve en el tablero, el ratón puede moverse a cualquier lugar del tablero, ya que es un ratón saltador. Para que usted pueda moverse en el tablero, el programa le debe preguntar cada vez, si se mueve a la izquierda, derecha, arriba o abajo. Si el ratón, y usted se encuentra en la misma posición del tablero, usted gana el juego y el programa termina.

	1	2	3	4	5
1	J				
2					
3			R		
4					
5					