1. Definición

Un arreglo (*array*) es un conjunto de datos **del mismo tipo** almacenados en una única variable a los cuales se puede acceder a través de uno o más índices, dependiendo de la dimensión del mismo. El arreglo tendrá tantos índices como dimensiones posea. Es una manera de tener indexados y/u ordenados los datos, agrupándolos por practicidad o necesidad, dependiendo el algoritmo a modelar. Los índices o posiciones del arreglo en `Python` siempre empiezan por el 0 (cero).

2. Unidimensionales (vectores)

2.1. Declaración y acceso

Los arreglos más sencillos son los que poseen un único índice o dimensión. A estos suele denominárselos **vectores**. Por ejemplo, un arreglo de 10 enteros se lo declara de la siguiente manera:

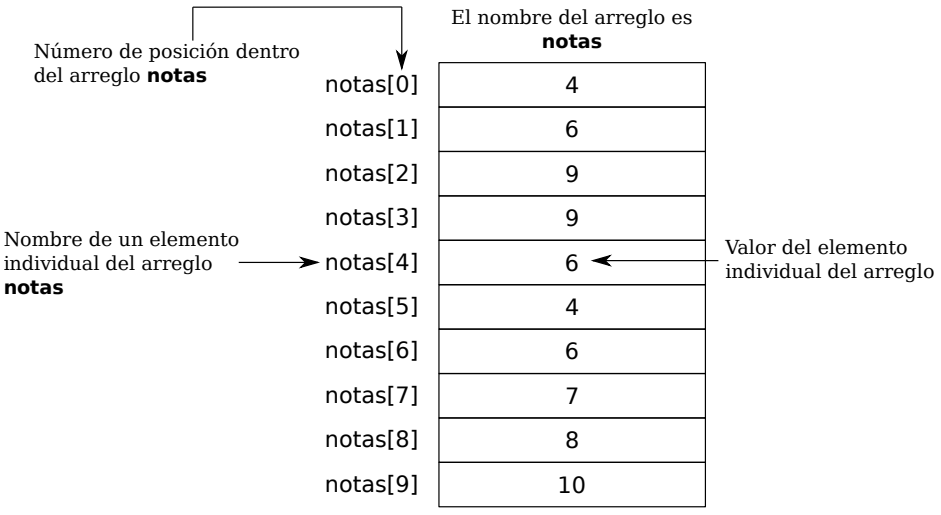
```
notas = [int() for n in range(10)]
print(f"Arreglo notas = {notas}")
```

Salida del programa:

```
Arreglo notas = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

En Python los *corchetes* `[]` instancian una **lista** de objetos. De esta manera, nosotros estamos iniciando un arreglo de 10 elementos con un valor inicial **`'0'`**. Y luego podemos modificar estos valores accediendo a cada una de las posiciones utilizando el índice correspondiente (del 0 al 9):

```
notas[0] = 4    # se almacena un 4 en la primera posición
notas[1] = 6    # se almacena un 6 en la segunda posición
notas[2] = 9    # se almacena un 9 en la tercera posición
# ...
notas[9] = 10   # se almacena un 10 en la última posición
```



Esto es tanto para almacenar un valor como para leerlo, por ejemplo, si se deseara mostrar alguno por pantalla se debe hacer:

```
print(f"La octaba nota es: {notas[7]}")
```

2.1.1. Indexación por medio de variables

No es necesario que el número sea escrito literal por el programador, el arreglo puede estar indexado por una variable. Por ejemplo, si se deseara mostrar todas las `notas`:

```
for i in range(0,10,1):
  print(f"La nota {i+1} es: {notas[i]}")
```

Es importante destacar que **`i` debe comenzar en 0** y terminar en el *valor correcto*, es decir, el último elemento del arreglo (en este caso 9). Ya que, de lo contrario, obtendremos un error conocido como `IndexError` y nuestro programa terminará abruptamente. La salida producida por este fragmento de código es la siguiente:

```
La nota 1 es: 4
La nota 2 es: 6
La nota 3 es: 9
...
La nota 10 es: 10
```

2.1.2. Mutabilidad

Como podemos observar, las listas son variables que soportan la asignación, es decir que son *mutables*. Y debemos tener en cuenta que cuando asignamos esta variable a otra, es decir, que apunten al mismo objeto en memoria, esto producirá un cambio en la otra:

```
ref_notas = notas  # apuntan al mismo objeto en memoria
ref_notas[2] = 10
print(f"La 3er nota es: {notas[2]}") # se muestra un 10
```

2.1.3. `for` para recorrer listas

Cabe destacar que en `Python` podemos recorrer una lista sin la necesidad de indicar un rango:

```
for n in notas: # para cada 'n' en notas:
  print(f"[{n}]",end=' ')

print() # línea vacía
```

En la variable `n` se irán copiando consecutivamente los valores almacenados en `notas`, se observa que no hace falta especificar el largo del arreglo, eso es deducido automáticamente. Una posible salida de este bucle es:

```
[4] [6] [9] [9] [6] [4] [6] [7] [8] [10]
```

La ventaja de esta técnica es que permite en una sintaxis muy abreviada recorrer todos los valores del vector, pero no es útil si se desea trabajar con los índices, ya que no tenemos la variable de control '`i`' para saber en qué posición nos encontramos. Los algoritmos que requieran utilizar el índice del arreglo, como por ejemplo, los de ordenamiento que serán explicados más adelante requerirán del `range()`.

Esta técnica es útil para mostrar, pero si lo que se desea es cargar el arreglo con valores, esto debe hacerse con un `range()`, pudiendo aprovechar la función `len()` la cual nos da la dimensión del mismo:

```
for i in range(len(notas)):
  notas[i] = random.randint(1,10)
```

2.2. Inicialización

Lo más habitual en `Python` inicializar la lista directamente, simplemente enumerando sus valores:

```
materias = ["Literatura", "Matemática","Ed. Físca","Física",
            "Historia", "Geografía", "Programación","Inglés",
            "Alemán", "Biología"]
```

Es posible recorrer más de un vector al mismo tiempo, en especial si estos están relacionados, como se observa en el Programa 1. Donde se vinculan los nombres de las materias con sus respectivas notas.

Programa 1: Boletín escolar

```
''' Funciones de entrada segura '''
def pedir_entero(msj:str) -> int:
  while True:
    try:
      x = int(input(msj))
      return x
    except ValueError:
      print("Error de ingreso, intente nuevamente...")

def pedir_entero_entre(msj:str, vmin:int, vmax:int) -> int:
  if vmin > vmax:            # si valor mínimo > v.máximo
    vmin, vmax = vmax, vmin  # se intercambian los valores

  while True:
    x = pedir_entero(msj)
    if(x >= vmin and x <= vmax):
      return x
    else:
      print("Error de ingreso, intente nuevamente...")


```

```python
''' Cuerpo principal del programa '''
def main():
  materias = ["Literatura","Matemática", "Ed. Físca","Física",
              "Historia","Geografía","Programación","Inglés",
              "Alemán","Biología"]
  # tantas notas como materias
  notas = [int() for i in range(len(materias))]

  for i in range(len(materias)):
    print(f"\n* Materia: {materias[i]} *")
    notas[i]=pedir_entero_entre("Nota (del 1 al 10):", 1, 10);

  # Se calcula el promedio:
  promedio = 0
  for nota in notas: # Para cada 'nota' en "notas"
    promedio += nota # Se acumulan las notas

  promedio /= len(materias) # se divide por cantidad de materias
  print(f"\n\nEl promedio general es: {promedio}")

main()
```

2.2.1. Copiar vectores

Como ya se ha analizado previamente, en Python, las variables almacenan una referencia al objeto en memoria. Por lo tanto si se hiciera una simple asignación (=), estaríamos referenciando la misma lista en memoria. Si se quisiera una copia del arreglo, es decir una lista con los mismo valores se podría optar por alguna de las siguientes opciones:

```python
# Vector/arreglo original
vec = [1,2,3,4,5,6,7,8,9,10]

# método 1:
# genera un nuevo objeto al sumarle otra lista vacía
cpv = vec+[]

# método 2:
# genera un nuevo objeto a partir del vector original
cpv = list(vec)
```

2.3. Arreglos como parámetros de una función

Como se indicó previamente, las listas son tipos de dato mutables, eso quiere decir que sus estados pueden cambiar dentro de otros ámbitos. Por ejemplo en una función que lo recibe como parámentro:

Programa 2: Arreglo como parámetro

```python
import random

def promediar(valores : list[int]) -> float:
  resultado = 0
  for i in range(len(valores)):
    resultado += valores[i]
  return resultado/len(valores)

def main():
  CANT_NOTAS = 10   # cantidad de notas
  MAX_NOTA   = 10   # nota máxima

  notas = [int() for i in range(CANT_NOTAS)]

  # Se generan CANT_NOTAS al azar entre 1 y MAX_NOTA
  for n in range(CANT_NOTAS):
    notas[n] = random.randint(1,MAX_NOTA)

  # Se muestran las notas generadas y se promedian:
  for n in notas:
    print(f"[{n}]", end=' ')

  print(f"\n\nEl promedio es: {promediar(notas)}")

main()
```

2.4. Strings como arreglo de caracteres

Los `str` en Python pueden recorrerse como si se tratara de arreglos letra a letra. Es decir que podemos acceder a cada una de ellas por separado utilizando los corchetes `'[]'`.

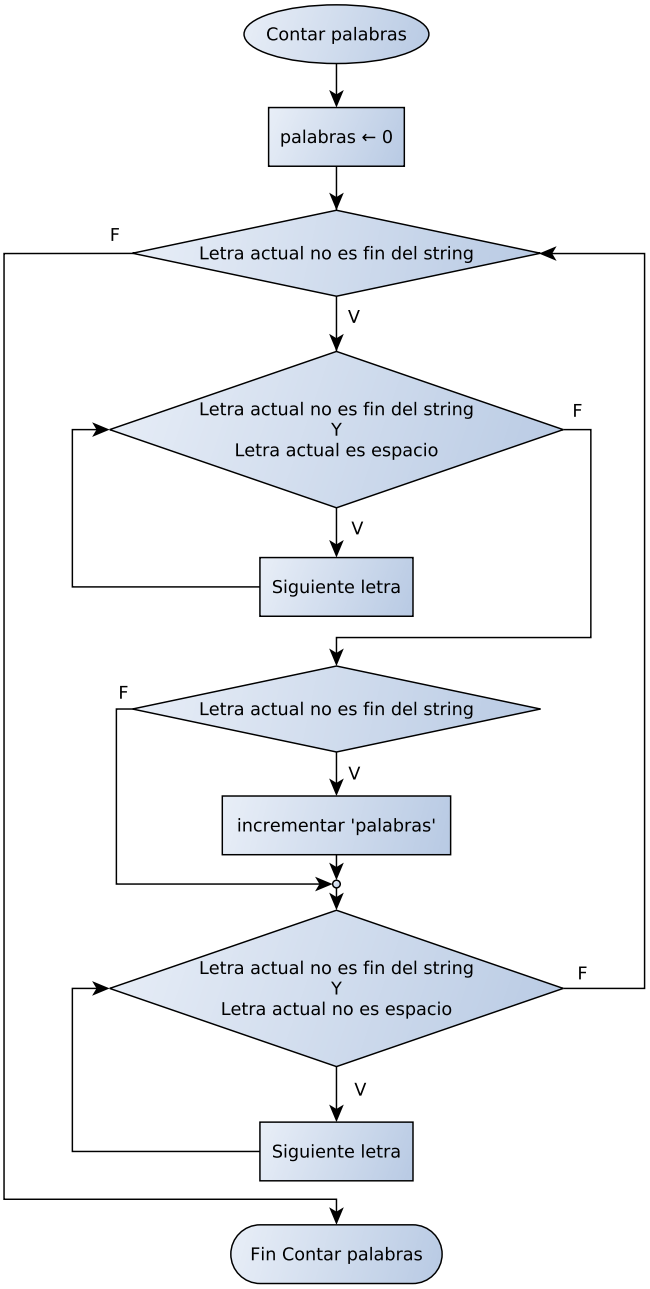
```python
texto = "Hola!"
for l in texto:
  print(f"[{l}]", end='')

print() # línea vacía

# Aleternativa con range():
for i in range(len(texto)):
  print(f"[{texto[i]}]", end='')

print() # línea vacía
```

```
[H][o][l][a][!]
```

2.4.1. Algoritmos con strings

El manejo de *strings* es un tema muy habitual en los sistemas, porque gran parte de la información está en este formato o, al menos, al momento de representar la información para los usuarios es imperativo. El usuario siempre interacturá con una interfaz que muestra y lee cadenas de texto.

Se analizarán algunos ejemplos de algoritmos con strings. Para empezar se desarrollará un algoritmo para contar palabras en un `string`. El mismo consiste en leer letra a letra el texto filtrando los espacios hasta encontrar una carácter que no sea espacio (es decir, una palabra), luego se recorren todas las letras de la palabra hasta encontrar un nuevo espacio y se repite hasta alcanzar el final del texto.



Programa 3: Contar palabras en un string

```
1  def contar_palabras(texto:str) -> int:
2    l = 0          # índice de la letra actual
3    palabras = 0   # cantidad de palabras
4    cant_letras = len(texto)
5
6    while l < cant_letras:
7      # se filtran todos los espacios:
8      while l < cant_letras and texto[l].isspace():
9        l += 1
10     # Si no es el final del string, es una palabra nueva
11     if l < cant_letras:
12       palabras += 1
13     # Se busca el próximo espacio (fin de la palabra actual)
14     while l < cant_letras and not texto[l].isspace():
15       l += 1
16
17   return palabras
18
19 def main():
20   texto = "  Esto   es  un   texto   de   prueba  "
21   print(f'El texto es "{texto}"\n'
22         f'Y tiene {contar_palabras(texto)} palabras.')
23
24 main()
```

```
El texto es "  Esto   es  un   texto   de   prueba  "
Y tiene 6 palabras.
```

En este ejemplo, el índice `l` dentro de la función `contar_palabras()` es incrementado mientras haya espacios dentro del *string* consulandolo a través de la función estándar `isspace()` o se haya alcanzado el final del mismo (líneas 8 y 9). Si no se alcanzó el final, eso quiere decir que hay una palabra nueva y se incrementa el contador de palabras (líneas 12 y 13). Finalmente se busca el siguiente espacio, es decir, el final de la palabra encontrada incrementando `l` (líneas 16 y 17), para luego seguir recorriendo el *string* repitiendo las acciones hasta el final del mismo.

A continuación se desarrolla un algoritmo que invierte un texto. Esto se logra recorriendo letra a letra el texto de atrás hacia adelante almacenándolo en otro `string` llamado '`inv`'. Debemos tener en cuenta que los strings en Python son inmutables, por lo tanto, generaremos un nuevo objeto cada vez que le agreguemos un nuevo carácter a `inv`, quedando finalmente la inversión de la cadena.

Programa 4: Invertir un string

```
1  def invertir(texto:str) -> str:
2    inv = "" # string inicialmente vacío
3    fin = len(texto)
4    for i in range(fin):
5      inv += texto[fin-1-i]
6    return inv
7
```

```
def main():
  texto = "Hola"
  print(f'El original es: "{texto}"\n'
        f'Invertido: "{invertir(texto)}"')

main()
```

```
El original es: "Hola"
Invertido: "aloH"
```

En este algoritmo se inicializa la variable **inv** con un string vacío. Luego se almacena la longitud de la cadena en la variable **fin** utilizando la función **len()**. Por último se recorrer la cadena concatenando en **inv** la última letra del parámetro **texto** utilizando el índice **fin-1**, y luego al ir incrementándose **i**, el indice entre corchetes ira descendiendo hasta llegar a la primera posición (**texto[0]**). En cada concatenación se genera un **str** nuevo cada vez con una letra más en la secuencia que se muestra a continuación:

texto←"Hola"			fin←4	
i	fin-1-i	inv[i]	texto[fin-1-i]	char
0	3	inv[0]	← texto[3]	'a'
1	2	inv[1]	← texto[2]	'l'
2	1	inv[2]	← texto[1]	'o'
3	0	inv[3]	← texto[0]	'H'

Por último, se analiza un algoritmo que cuenta vocales en un texto dado. Esto se realizará con simples comparaciones y aprovechando que Python utiliza codificación **UTF-8** por *default*.

Programa 5: Contar las vocales en un texto

```
def contar_vocales(texto:str) -> int:
  VOCALES = "AEIOUÁÉÍÓÚ"
  cant_vocales = 0
  i = 0
  while i < len(texto):
    for v in VOCALES:
      # Se compara el carácter en mayúscula
      if texto[i].upper() == v:
        cant_vocales += 1
    i += 1
  return cant_vocales

def main():
  texto = "  Ésto  es  un  texto  de  prueba  ";
  print(f'El texto es "{texto}"\n'
        f'Y tiene {contar_vocales(texto)} vocales.')

main()
```

```
El texto es "  Ésto  es  un  texto  de  prueba  "
Y tiene 10 vocales.
```

En este caso se utiliza una función estándar `upper()` que transforma el string en mayúscula, para luego ser comparada con cada vocal en el arreglo `vocales` (línea 8). En caso de coincidir se incrementa la cuenta de vocales llevada en `cant_vocales` (línea 9) y, finalmente, se pasa a la siguiente letra (línea 10) hasta llegar al final del texto.

3. Multidimensionales (matrices)

La primera aproximación que podemos hacer a los arreglos con más de una dimensión es entenderlos como un vector de vectores. Supongamos entonces:

```
# vector de 3 posiciones
# cuyos elementos son vectores de 5 posiciones:
matriz = [[ 2, 3, 6,-1, 6],
          [ 7,-2,-7, 9,10],
          [-9,-3, 0,22,-5]]
```

Se observa además, que al inicializar el arreglo de 2 dimensiones, cada uno de sus elementos son arreglos que también deben estar entre `[]`. Si se deseara acceder a un elemento del arreglo debemos indicar ambos índices.

Representación del arreglo **matriz**

	[0]	[1]	[2]	[3]	[4]
[0]	2	3	6	-1	6

	[0]	[1]	[2]	[3]	[4]
[1]	7	-2	-7	9	10

	[0]	[1]	[2]	[3]	[4]
[2]	-9	-3	0	22	-5

Por lo tanto si se quisiera acceder a una de las posiciones se debe indicar de la siguiente manera:

```
print(f"La posición (2, 4) de la matriz es: {matriz[1][3]}")
```

```
La posición (2, 4) de la matriz es: 9
```

3.1. Recorrer un arreglo multidimensional

Para poder recorrer un arreglo usualmente es necesario utilizar tantos bucles como dimensiones tenga. Por lo tanto, para un arreglo de dos dimensiones, serán necesarios dos bucles y dos índices:

```
FILAS    = 3
COLUMNAS = 5

matriz = [[int() for c in range(COLUMNAS)] for f in range(FILAS)]

# Para ingresar datos:
for fila in range(FILAS):
  for col in range(COLUMNAS):
    matriz[fila][col] = int(input("Ingrese posición "
                                  f"({fila+1}, {col+1}): "))

# Para mostrar los datos:
for fila in range(FILAS):
  print("[ ",end='')
  for col in range(COLUMNAS):
    print(f"{matriz[fila][col]}",end=', ')
  print("\b\b ]")
```

Alternativamente se puede usar para mostrar:

```
# Para cada fila en la matriz:
for fila in matriz:
  print("[ ",end='')
  # para cada elemento en la fila:
  for elem in fila:
    print(f"{elem}",end=', ')
  print("\b\b ]")
```

A continuación se expone un ejemplo concreto de utilización de arreglos multidimensionales, extendiendo el Programa 1 incluyendo notas trimestrales para cada materia y utilizando las funciones anteriormente expuestas para ingresar las notas y calcular el promedio de cada materia.

Programa 6: Boletin anual trimestral

```
''' Funciones ya implementadas (no se muestran) '''
def pedir_entero(msj:str) -> int:
  # ...
def pedir_entero_entre(msj:str, vmin:int, vmax:int) -> int:
  # ...
def promediar(valores:list[int]) -> float:
  # ...

def main():
  # para Trimestres son 3 notas
  CANT_NOTAS = 3

  materias = ["Literatura","Matemática","Ed. Físca","Física",
              "Historia","Geografía","Programación","Inglés",
              "Alemán","Biología"]

```

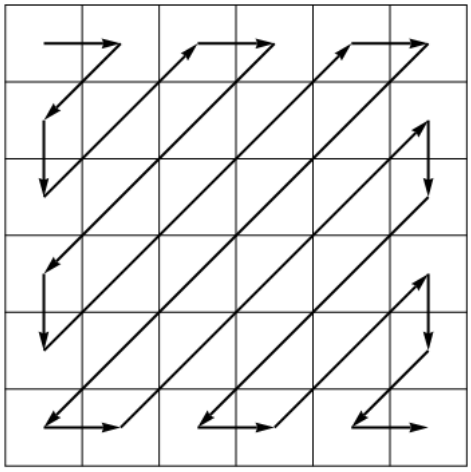
```
17    # Se agregan las 3 notas a cada materia:
18    boletin = [[int() for n in range(CANT_NOTAS)]
19                      for e in range(len(materias))]
20
21    # Para ingresar datos:
22    for materia in range(len(materias)):
23      texto = f"Ingrese la nota de {materias[materia]}\n"
24      for nota in range(CANT_NOTAS):
25        boletin[materia][nota] = pedir_entero_entre(
26                              texto+f"Del Trimestre {nota+1}: ",1,10)
27
28    print() # línea vacía
29
30    # Se muesta el promedio de cada materia:
31    for i in range(len(materias)):
32      print(f"Promedio de {materias[i]}: {promediar(boletin[i])}")
33
34  main()
```

Si se deseara podría agregarse a los estudiantes, así se haría presente una nueva dimensión en el arreglo:

```
estudiantes = ["María","Pedro","Juan","Sofía","Malena"]

boletines = [[[int() for n in range(CANT_NOTAS)]
                    for m in range(len(materias))]
                    for e in range(len(estudiantes))]

# Para recorrer y cargar valores:
for estudiante in range(len(estudiantes)):
  for materia in range(len(materias)):
    for nota in range(CANT_NOTAS):
      boletines[estudiante][materia][nota] = pedir_entero_entre(
```

Esto se interpreta como que hay 5 estudiantes y cada uno tienen 10 materias, y cada materia 3 notas cuyo formato es un entero (`int`). Siempre se está hablando de una nota que es la información almacenada, pero está estructurada a través de los índices y relacionada con otros datos (`materias[]` y `estudiantes[]`).

Como aclaración final, aquí se recorrieron los arreglos de forma lineal, pero existen motivos algorítmicos para desear recorrer las matrices de forma parcial o completa en formas *no-lineales*, por ejemplo en *zig-zag*, por las diagonales, en espiral, etc. Esto es aplicado en diferentes algoritmos de compresión de datos y otros cálculos más avanzados que exceden el contenido de este curso.



Recorrido de matiz en Zig-Zag

4. Ejercitación

Codifique los siguientes programas utilizando arreglos y funciones en `Python`.

1. Pida ingresar 10 números reales y almacénelos en un arreglo. Luego sume los elementos que se encuentran en posiciones pares, reste los que se encuentran en posiciones impares, muestre ambos valores.

2. Pida ingresar 10 números reales y calcule el promedio de los mismos, luego muestre cuales están por encima de ese promedio.

3. Desarrolle un software que pida 10 enteros distintos, en caso de ingresar un valor repetido debe volver a pedir otro hasta completar los 10. Luego, muestre los números ingresados en pantalla.

4. Genere 10 números aleatorios entre -10 y 10 y guárdelos en un arreglo. Muéstrelos y determine cual es el mayor. La función debe estar definida para enteros.

5. Pida ingresar 10 valores numéricos y almacénelos en un arreglo. Busque dentro del arreglo el mayor y el menor valor e intercambie sus posiciones. Por último muestre el arreglo.

6. Desarrolle un software que pida ingresar el nombre completo de 5 estudiantes y sus promedios. Luego **muestre el nombre** del estudiante con mejor promedio.

7. Desarrolle un software que permita tirar 5 dados de 6 caras (al azar) y permita cambiar el valor de hasta 3 de ellos a elección indicando la posición de los dados que se desean cambiar (del 1 al 5).

8. Desarrolle un software donde pueda ingresar el nombre de un restaurante y 10 valoraciones entre 1 y 5 para cada uno. Muestre el nombre del restaurante y su puntuación promedio redondeado a 1 decimal.

9. Extienda el Programa 6 de la pág. 10, agregando una dimensión más al arreglo para 5 estudiantes, los mismos deben ser ingresados por la terminal y luego de mostrar los promedios trimestrales de cada materia de cada estudiante indique el nombre del que tenga mejor promedio general (promedio de los promedios).

10. Desarrolle un software que genere una matriz de enteros de 9x9 donde los valores se generen aleatoriamente del 1 al 9 y no puedan repetirse en ninguna fila o columna.