Hacer en Python los siguientes ejercicios

Arrays unidimensionales y matrices con funciones

Ejercicio 1.- Crear un array unidimensional de 20 elementos con nombres de personas. Visualizar los elementos de la lista debiendo ir cada uno en una fila distinta.

```
 Solicitar al usuario el número de nombres a introducir
n = int (input ("¿Cuántos nombres deseas introducir? "))

 Crear un array unidimensional vacío para almacenar los nombres
nombres = []

print (f" Por favor, introduce {n} nombres:")

 Solicitar al usuario que introduzca los nombres uno por uno
for i in range(n):
    nombre = input (f" Nombre {i + 1}: ")
    nombres.append(nombre)

 Visualizar los elementos de la lista, uno por línea
print ("\n Lista de nombres introducidos:")
for nombre in nombres:
    print(nombre)
```

Solución con funciones.

```
def crearLista(n):
        print(f"Escriba los {n} nombres:")
        nombres=[]
        for i in range(n):
                nombre=input(f"digite nombre {i+1}: ")
                nombres.append(nombre)
return nombres

def imprimirListaFila(nombres):
        for nombre in nombres:
        print(nombre)

if __name__=='__main__':
        n=int(input("digita cuantos elementos deseas crear: "))
        nombres=crearLista(n)
```

```
imprimirListaFila(nombres)
```

Ejercicio 2.-Hacer un programa que lea las calificaciones de un alumno en 10 asignaturas, las almacene en un vector y calcule e imprima su media.

```
 Inicializar la lista para almacenar las notas
notas = []
suma = 0  Inicializar la suma de notas
for i in range (1, 11):  Solicitar al usuario que ingrese 10 notas
    nota = float (input (f" Nota {i}: "))
    notas.append(nota)
suma = sum(notas)  Calcular la suma de todas las notas con sum ()
media = suma / 10  Calcular la media de las notas
print (f" Nota media: {media}")  Imprimir la nota media
```

Ejercicio 3.- Escribe un programa en Python que realice las siguientes acciones:

1. Inicialice una lista vacía llamada notas para almacenar las calificaciones de los estudiantes.
2. Solicite al usuario que ingrese 10 notas. Cada nota debe ser un número decimal y se almacenará en la lista notas.
3. Solicite al usuario que ingrese una nota específica que desea buscar en la lista de notas.
4. Busque todas las posiciones en la lista notas donde se encuentra la nota especificada por el usuario y almacene estas posiciones en una lista llamada posiciones.
5. Verifique si la nota especificada se encontró en la lista notas.
6. Si se encontraron posiciones, imprima las posiciones (basadas en 1, no en 0) donde se encuentra la nota.
7. Si la nota no se encontró en la lista notas, imprima un mensaje indicando que la nota no se encontró.

Solución:

```
notas = []  Inicializar la lista para almacenar las notas
 Solicitar al usuario que ingrese 10 notas

for i in range (10):
    nota = float (input (f" Nota {i}: "))
    notas.append(nota)

 Solicitar al usuario la nota a buscar en el vector
nota_buscar = float (input ("Ingrese la nota que desea buscar: "))
```

```
posiciones = []  Inicializar lista para guardar las posiciones donde se encuentra la nota

for i, n in enumerate (notas):  Buscar la nota y su posición en el vector
    if n == nota_buscar:  si es la nota buscada guarda su posición en posiciones.
        posiciones.append(i)

if posiciones:  si existe posición mostrar la posición
    print (f" La nota {nota_buscar} se encontró en la(s) posición(es): {i}")
else:
    print ("La nota no se encontró en el vector.")
```

Explicación:

- La función enumerate toma una lista (en este caso, notas) y devuelve un iterador que produce pares de valores (índice, elemento).
- Por ejemplo, si notas es [3.5, 4.0, 2.8, 3.5], enumerate(notas) generará los pares (0, 3.5), (1, 4.0), (2, 2.8), (3, 3.5).

Ejercicio con funciones:

```
def ingresar_notas(cantidad):
    notas = []
    for i in range(cantidad):
        nota = float(input(f"Nota {i + 1}: "))
        notas.append(nota)
    return notas

def buscar_nota(notas, nota_buscar):
    posiciones = []
    for i, n in enumerate(notas):
        if n == nota_buscar:
            posiciones.append(i)
    return posiciones

def main():
    cantidad_notas = 10
    notas = ingresar_notas(cantidad_notas)

    nota_buscar = float(input("Ingrese la nota que desea buscar: "))
    posiciones = buscar_nota(notas, nota_buscar)

    if posiciones:
```

```
        print(f"La nota {nota_buscar} se encontró en la(s) posición(es): {posiciones}")
    else:
        print("La nota no se encontró en el vector.")

if __name__ == "__main__":
    main()
```

Ejercicio 4.- crea una tupla con la información de un participante del torneo de ajedrez y luego imprime cada uno de los elementos de la tupla con una descripción adecuada:

```
# Crear una tupla con la información del participante
participante = ("John Doe", 30, "Canadá", 2400)

# Imprimir cada elemento de la tupla con una descripción
print (f"Nombre del participante: {participante[0]}")
print (f"Edad: {participante[1]}")
print (f"País: {participante[2]}")
print (f"Calificación Elo: {participante[3]}")
```

Ejercicio 5.- Se te ha asignado la tarea de desarrollar un programa que cree y muestre los elementos de una matriz bidimensional de dimensiones 4x5, es decir, 4 filas y 5 columnas. Cada elemento de la matriz debe ser un número entero generado aleatoriamente entre 1 y 100, inclusivo. Una vez generada, la matriz debe ser mostrada en pantalla, elemento por elemento, junto con sus índices correspondientes para facilitar su identificación.

```
import random
mat = []  Crear e inicializar la matriz de dimensiones 4x5
# Llenar la matriz con números aleatorios
for i in range(4):   no hay necesidad de inicializar i, hace 4 filas
    fila = []   Crear una nueva fila
    for j in range(5):   no hay necesidad de iniciar j, hace 5 columnas por fila
        numero_aleatorio = random.randint(1, 100)   Número aleatorio entre 1 y 100
        fila.append(numero_aleatorio)   Añadir el número a la fila actual
    mat.append(fila)   Añadir la fila completa a la matriz A

# Mostrar la matriz y sus índices
for i in range(4):
    for j in range(5):
        print (f" Índice [{i}, {j}] = {mat[i][j]}")
    print ("")   Imprimir una línea vacía después de cada fila para mejor legibilidad

# Imprimir la matriz
print("Matriz generada:")
for fila in mat:
```

```
    for elemento in fila:
        print(f"{elemento:3}", end=" ")   Imprimir c/u con formato y un espacio
    print()   Imprimir una nueva línea después de cada fila
```

Explicación del imprimir: print(f"{elemento:3}", end=" "): Este comando imprime cada elemento de la fila.

1. **f"{elemento:3}":** Esto es una f-string (cadena de formato) en Python, que permite insertar variables dentro de una cadena. En este caso, elemento es la variable que se está insertando.
2. **:3** dentro de la f-string especifica que el valor de elemento debe ocupar al menos 3 espacios de ancho. Esto ayuda a alinear los números en columnas, especialmente cuando tienen diferentes cantidades de dígitos (por ejemplo, números de 1 dígito, 2 dígitos, etc.).
3. **end=" ":** El argumento end de la función print especifica lo que se debe imprimir al final. Por defecto, print añade una nueva línea (\n) al final de cada llamada. Aquí, end=" " cambia este comportamiento para que al final de cada impresión se añada un espacio en lugar de una nueva línea. Esto asegura que todos los elementos de una fila se impriman en la misma línea, separados por espacios.

Ejercicio 6.- Se requiere el desarrollo de un programa que demuestre la creación y manipulación de matrices en la programación, específicamente enfocándose en el concepto de la transposición de matrices. Para este fin, el programa debe realizar las siguientes tareas:

a) Generar una Matriz Aleatoria: El programa debe inicialmente crear una matriz A de dimensiones 4x5, es decir, compuesta por 4 filas y 5 columnas. Cada elemento de esta matriz debe ser un número entero aleatorio generado dentro del rango de 1 a 100.

b) Transponer la Matriz: Posteriormente, el programa debe calcular la matriz transpuesta de A, denominada B. Dado que A es una matriz de 4x5, B será una matriz de 5x4. La transposición implica que el elemento en la posición (i, j) en A será trasladado a la posición (j, i) en B.

c) Visualización de las Matrices: Después de generar la matriz original y su transpuesta, el programa debe mostrar ambas matrices en la pantalla. Primero se debe mostrar toda la matriz original A, seguida por la matriz transpuesta B. Cada fila de la matriz se debe imprimir en una línea separada para facilitar la visualización.

```
import random

 a) Generar una Matriz Aleatoria
 Crear e inicializar la matriz A de dimensiones 4x5
A = []
for i in range(4):
```

```
    fila = []
    for j in range(5):
        numero_aleatorio = random.randint(1, 100)   Número aleatorio entre 1 y 100
        fila.append(numero_aleatorio)
    A.append(fila)

 b) Transponer la Matriz
 Crear la matriz B de dimensiones 5x4
B = []
for j in range(5):
    fila = []
    for i in range(4):
        fila.append(A[i][j])
    B.append(fila)

 c) Visualización de las Matrices
 Imprimir la matriz original A
print("Matriz A (Original):")
for fila in A:
    for elemento in fila:
        print(f"{elemento:3}", end=" ")
    print()

 Imprimir la matriz transpuesta B
print("\nMatriz B (Transpuesta):")
for fila in B:
    for elemento in fila:
        print(f"{elemento:3}", end=" ")
    print()
```

Ejercicio 7.- Desarrolla un programa para registrar notas de estudiantes en múltiples cursos, solicitando primero la cantidad de cursos y estudiantes por curso, y luego las notas individuales para almacenarlas en una matriz.

```
 Inicializar la lista para almacenar las notas de todos los cursos
notas_cursos = []

 Solicitar la cantidad de cursos
cantidad_cursos = int(input("Ingrese la cantidad de cursos: "))

 Solicitar la cantidad de estudiantes por curso
cantidad_estudiantes = int(input("Ingrese la cantidad de estudiantes por curso: "))

 Recoger las notas para cada curso y estudiante
for curso in range(cantidad_cursos):
    print(f"\nCurso {curso + 1}:")
```

```
    notas_curso = []
    for estudiante in range(cantidad_estudiantes):
        nota = float(input(f"  Ingrese la nota del estudiante {estudiante + 1}: "))
        notas_curso.append(nota)
    notas_cursos.append(notas_curso)

 Imprimir la matriz de notas
print("\nMatriz de Notas:")
for curso in range(cantidad_cursos):
    print(f"Curso {curso + 1}: ", end="")
    for estudiante in range(cantidad_estudiantes):
        print(f"{notas_cursos[curso][estudiante]:.2f}", end=" ")
    print()
```

Taller de Python: Funciones Sencillas, Funciones con Parámetros y Ejercicios de Matrices

Ejercicio 8: Función Sencilla para Sumar Números Pares

Escribe una función que sume todos los números pares en una lista.

```
def sumar_pares(lista):
    return sum(num for num in lista if num % 2 == 0)

if __name__ == '__main__':
    lista = [1, 2, 3, 4, 5, 6]
    suma_pares = sumar_pares(lista)
    print(f"La suma de los elementos pares es: {suma_pares}")
```

Ejercicio 9: Función con Parámetros para Contar Ocurrencias

Escribe una función que cuente cuántas veces aparece un nombre específico en una lista.

```
def contar_ocurrencias(lista, nombre):
    return lista.count(nombre)
```

```
if __name__ == '__main__':
    empleados = ["Ana", "Luis", "Carlos", "Ana", "María", "Ana"]
    nombre = "Ana"
    contador = contar_ocurrencias(empleados, nombre)
    print(f"El nombre '{nombre}' aparece {contador} veces en la lista de empleados.")
```

Ejercicio 10: Función con Parámetros para Eliminar Duplicados

Escribe una función que elimine los duplicados de una lista.

```
def eliminar_duplicados(lista):
    return list(set(lista))

if __name__ == '__main__':
    salarios = [3000, 2000, 4000, 3000, 2500, 4000]
    salarios_sin_duplicados = eliminar_duplicados(salarios)
    print(f"Lista de salarios sin duplicados: {salarios_sin_duplicados}")
```

Ejercicio 11: Función Sencilla para Calcular Factoriales

Escribe una función que genere una lista de factoriales de los números del 1 al 5.

```
import math

def calcular_factoriales(n):
```

```
    return [math.factorial(i) for i in range(1, n+1)]

if __name__ == '__main__':
    factoriales = calcular_factoriales(5)
    print(f"Lista de factoriales: {factoriales}")
```

Ejercicio 12: Función con Parámetros para Filtrar Números Mayores que un Valor Dado

Escribe una función que filtre los números mayores que un valor dado en una lista.

```
def filtrar_mayores(lista, valor):
    return [num for num in lista if num > valor]

if __name__ == '__main__':
    lista = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
    valor = 5
    mayores_que_valor = filtrar_mayores(lista, valor)
    print(f"Números mayores que {valor}: {mayores_que_valor}")
```

Ejercicio 13: Función para Verificar Palíndromos

Escribe una función que verifique si una palabra es un palíndromo.

```
def es_palindromo(palabra):
    palabra = palabra.lower()
    return palabra == palabra[::-1]
```

```
if __name__ == '__main__':
    palabra = "Reconocer"
    resultado = es_palindromo(palabra)
    print(f"¿La palabra '{palabra}' es un palíndromo? {resultado}")
```

Ejercicio 14: Función para Invertir una Cadena

Escribe una función que invierta una cadena de texto.

```
def invertir_cadena(cadena):
    return cadena[::-1]

if __name__ == '__main__':
    cadena = "Python"
    cadena_invertida = invertir_cadena(cadena)
    print(f"Cadena invertida: {cadena_invertida}")
```

Ejercicio 15: Función para Filtrar Palabras que Contengan una Subcadena

Escribe una función que filtre una lista de palabras y devuelva solo las palabras que contienen una subcadena específica.

```
def filtrar_palabras_subcadena(palabras, subcadena):
    return [palabra for palabra in palabras if subcadena in palabra]

if __name__ == '__main__':
    palabras = ["hola", "mundo", "holanda", "python", "holístico"]
    subcadena = "hol"
```

```
palabras_filtradas = filtrar_palabras_subcadena(palabras, subcadena)
print(f"Palabras que contienen '{subcadena}': {palabras_filtradas}")
```