MANUAL TECNICO

GESTOR DE NOTAS

registro de notas

```
def registronotacurso():
    n = int(input("ingrese la cantidad de cursos que desea registrar: "))
    for i in range(n):
        nombre = input(f"ingrese el nombre del curso {i + 1}: ").strip()
        if not nombre:
            print("el nombre del curso no puede estar vacío.")
            return
        try:
            nota = float(input("ingrese la nota obtenida (0-100): "))
            if nota < 0 or nota > 100:
                print("la nota debe estar entre 0 y 100.")
                return
        except ValueError:
            print("la nota debe ser un numero.")
            return
        historialcursos[nombre] = nota
        print(f"curso {nombre} registrado correctamente con nota {nota}.")
```

mostrar cursos

```
def mostrarcursos(historialcursos):
    if not historialcursos:
        print("no hay cursos registrados.")
    else:
        print("listado de cursos registrados:")
        for nombre, nota in historialcursos.items():
            print(f"{nombre} -> nota: {nota}")
```

promedio general

```
def promediogeneral(historialcursos):
    if not historialcursos:
        print("no hay cursos registrados para calcular el promedio.")
        return

    total = sum(historialcursos.values())
    promedio = total / len(historialcursos)
    print(f"promedio general calculado: {promedio:.2f}")
```

cursos aprobados cursos reprobados

```
def reproapro(historialcursos):
    aprobados = 0
    reprobados = 0
    for nota in historialcursos.values():
        if nota >= 60:
```

```
            aprobados += 1
        else:
            reprobados += 1
    print(f"total de cursos aprobados: {aprobados}")
    print(f"total de cursos reprobados: {reprobados}")
```

buscar curso por nombre, forma lineal

```
def busquedalineal_func(historialcursos, busquedalineal, historialgeneral):
    nombre = input("ingrese el nombre del curso a buscar: ").strip()
    if not nombre:
        print("este campo no puede estar vacio")
        return
    encontrado = False
    for curso in historialcursos:
        if curso.lower() == nombre.lower():
            print(f"curso encontrado: {curso} // nota: {historialcursos[curso]}")
            busquedalineal[curso] = busquedalineal.get(curso, 0) + 1
            print(f"este curso ha sido consultado {busquedalineal[curso]} veces.")
            historialgeneral.append(f"Se busco el curso {curso} - total de busquedas:
{busquedalineal[curso]}")
            encontrado = True
            break
    if not encontrado:
        print("curso no encontrado.")
        historialgeneral.append(f"busqueda fallida: curso {nombre} no encontrado.")
```

actualizar nombre del curso, nota de curso o ambos

```
def editarcursonota(historialcursos, historialgeneral):
    nombre = input("ingrese el nombre del curso que desea actualizar: ").strip()
    if not nombre:
        print("este campo no puede estar vacío.")
        return
    if nombre not in historialcursos:
        print(f"el curso '{nombre}' no esta registrado.")
        historialgeneral.append(f"intento fallido de actualizacion: curso {nombre} no
encontrado.")
        return
    cursoactual = nombre
    notaactual = historialcursos[cursoactual]
    while True:
        print(f"curso seleccionado: {cursoactual}  nota actual: {notaactual}")
        print("1. editar nombre del curso")
        print("2. editar nota del curso")
```

```
print("3. confirmar cambios")
try:
    opcion = int(input("seleccione una opcion: "))
except ValueError:
    print("ingrese una opción valida.")
    continue
if opcion == 1:
    nuevonombre = input("ingrese el nuevo nombre del curso: ").strip()
    if not nuevonombre:
        print("este campo no puede estar vacio.")
        continue
    if nuevonombre in historialcursos:
        print("ya existe un curso con ese nombre.")
        continue
    historialcursos[nuevonombre] = historialcursos.pop(cursoactual)
    historialgeneral.append(f"nombre actualizado: '{cursoactual}' -> {nuevonombre}'")
    cursoactual = nuevonombre
    print(f"nombre cambiado a {nuevonombre}.")
elif opcion == 2:
    try:
        nuevanota = float(input("ingrese la nueva nota (0-100): "))
        if nuevanota < 0 or nuevanota > 100:
            print("la nota debe estar entre 0 y 100.")
            continue
    except ValueError:
        print("la nota debe ser un numero.")
        continue
    historialcursos[cursoactual] = nuevanota
    notaactual = nuevanota
    historialgeneral.append(f"nota actualizada para {cursoactual}: {nuevanota}")
    print(f"nota cambiada a {nuevanota}.")
elif opcion == 3:
    print("actualizado... regresando al menu principal...")
    break
else:
    print("opción no valida.")
```

eliminación de curso

```
def eliminarcurso(historialcursos, historialgeneral):
    if not historialcursos:
        print("no hay cursos registrados para eliminar.")
        return
    nombre = input("ingrese el nombre del curso que desea eliminar: ").strip()
    if not nombre:
        print("este campo no puede estar vacio.")
        return
    curso_encontrado = None
    for curso in historialcursos:
        if curso.lower() == nombre.lower():
            curso_encontrado = curso
            break
    if curso_encontrado:
        confirmacion = input(f"seguro que desea eliminar {curso_encontrado} (s/n): ").lower()
        if confirmacion == 's':
            historialcursos.pop(curso_encontrado)
            historialgeneral.append(f"curso eliminado: {curso_encontrado}")
            print(f"Curso '{curso_encontrado}' eliminado correctamente")
        else:
            print("la eliminacion fue cancelada")
    else:
        print(f"el curso {nombre} no fue encontrado.")
        historialgeneral.append(f"intento fallido de eliminación: curso {nombre} no existe.")
```

ordenamiento por nota

```
def ordenar_por_nota(cursos, historial):
    if not cursos:
        print("No hay cursos para ordenar.")
        return

    print("1. De menor a mayor")
    print("2. De mayor a menor")
    opcion = int(input("Elige una opción: "))

    lista = list(cursos.items())  # Convertir el diccionario a lista de tuplas

    # Algoritmo burbuja clásico
    for i in range(len(lista)):
        for j in range(len(lista) - 1):
            if opcion == 1 and lista[j][1] > lista[j + 1][1]:
                lista[j], lista[j + 1] = lista[j + 1], lista[j]
            elif opcion == 2 and lista[j][1] < lista[j + 1][1]:
                lista[j], lista[j + 1] = lista[j + 1], lista[j]
```

```
    print("Cursos ordenados:")
    for nombre, nota in lista:
        print(f"{nombre} - Nota: {nota}")
        historial.append(f"{nombre} - Nota: {nota}")
```

ordenamiento por nombre

```
# 9. Ordenar por nombre (inserción)
def ordenar_por_nombre(cursos, historial):
    if not cursos:
        print("No hay cursos registrados para ordenar.")
        return
    print("¿Cómo deseas ordenar los nombres?")
    print("1. De A a Z")
    print("2. De Z a A")
    try:
        opcion = int(input("Seleccione una opción: "))
    except ValueError:
        print("Opción inválida.")
        return
    lista = list(cursos.items())
    # Ordenamiento por inserción
    for i in range(1, len(lista)):
        clave = lista[i]
        j = i - 1
        # Comparación de nombres (ignorando mayúsculas/minúsculas)
        while j >= 0 and (
            (opcion == 1 and clave[0].lower() < lista[j][0].lower()) or
            (opcion == 2 and clave[0].lower() > lista[j][0].lower())
        ):
            lista[j + 1] = lista[j]
            j -= 1
        lista[j + 1] = clave
    if opcion == 1:
        historial.append("Cursos ordenados por nombre (A-Z) con inserción")
    elif opcion == 2:
        historial.append("Cursos ordenados por nombre (Z-A) con inserción")
    print("Cursos ordenados por nombre:")
    for nombre, nota in lista:
        print(f" - {nombre} / Nota: {nota}")
        historial.append(f"Curso: {nombre} / Nota: {nota}")
```

buscar nombre de curso de forma binaria

```
def busqueda_binaria(cursos, historial):
    if not cursos:
        print("No hay cursos registrados.")
        return
    nombre = input("Ingrese el nombre del curso a buscar (binaria): ").strip()
    if not nombre:
        print("El nombre no puede estar vacío.")
        return
    nombres_ordenados = sorted(cursos.keys(), key=lambda x: x.lower())
    izquierda = 0
    derecha = len(nombres_ordenados) - 1
    encontrado = False
    while izquierda <= derecha:
        medio = (izquierda + derecha) // 2
        curso_medio = nombres_ordenados[medio]
        if curso_medio.lower() == nombre.lower():
            nota = cursos[curso_medio]
            print(f"Curso encontrado: {curso_medio} / Nota: {nota}")
            historial.append(f"Búsqueda binaria exitosa: '{curso_medio}' con nota {nota}")
            encontrado = True
            break
        elif nombre.lower() < curso_medio.lower():
            derecha = medio - 1
        else:
            izquierda = medio + 1
    if not encontrado:
        print("Curso no encontrado.")
        historial.append(f"Búsqueda binaria fallida: curso '{nombre}' no existe.")
```

simulación de cola de revisión

```
def simular_cola_revision(historialcursos):
    if not historialcursos:
        print("No hay cursos en el historial.")
        return
    nom2 = input("confirme su nombre para iniciar la revisión: ")
    if nom2.strip() == "":
        print("El nombre no puede estar vacío.")
        return
    else:
        nom2 == nom_pers.strip()
    if nom2 != nom_pers:
        print("El nombre no coincide con el registrado. Revisión cancelada.")
        return
    for curso, nota in historialcursos.items():
```

```
    print(f"Revisando curso: {curso} / Nota: {nota}")
print(f"Hola {nom2}, comenzando la revisión de cursos...")
print("Simulando cola de revisión...")
print("Revisión de todos los cursos completada.")
print(f"Gracias por tu paciencia, {nom2}. Revisión finalizada.")
print("regresando al menu principal...")
```

historial de cambios y busquedas.

```
def mostrarhistorial(historialgeneral):
    if not historialgeneral:
        print("no hay historial de acciones.")
    else:
        print("historial de acciones realizadas:")
        for evento in historialgeneral:
            print(f"- {evento}")
```

INIALIZACIÓN Y MENÚ

```
print("bienvenido al sistema de registro de notas")
nom_pers = input("Ingrese su nombre para empezar: ")
if nom_pers == "":
    print("El nombre no puede estar vacío. Reinicie el programa.")
    exit()
print("Nombre registrado, ¡HERE WE GO!")
while True:
    print("Menú principal:")
    print("1. Agregar curso")
    print("2. Mostrar cursos")
    print("3. Calcular promedio")
    print("4. Conteo de cursos aprobados y reprobados")
    print("5. Buscar curso por nombre (lineal)")
    print("6. Actualizar o editar nombre del curso/nota")
    print("7. Eliminar curso")
    print("8. Ordenar por nota")
    print("9. Ordenar por nombre")
    print("10. Buscar curso por nombre (binaria)")
    print("11. Simular cola de revisión")
    print("12. Mostrar historial de cambios")
    print("13. Salir")
```

```
try:
    opcion = int(input("Ingrese el número de la opción: "))
except ValueError:
    print("Por favor, ingrese un número válido.")
    continue
if opcion == 1:
    registronotacurso()
elif opcion == 2:
    mostrarcursos(historialcursos)
elif opcion == 3:
    promediogeneral(historialcursos)
elif opcion == 4:
    reproapro(historialcursos)
elif opcion == 5:
    busquedalineal_func(historialcursos, busquedalineal, historialgeneral)
elif opcion == 6:
    editarcursonota(historialcursos, historialgeneral)
elif opcion == 7:
    eliminarcurso(historialcursos, historialgeneral)
elif opcion == 8:
    ordenar_por_nota(cursos=historialcursos, historial=historialgeneral)
elif opcion == 9:
    ordenar_por_nombre(cursos=historialcursos, historial=historialgeneral)
elif opcion == 10:
    busqueda_binaria(cursos=historialcursos, historial=historialgeneral)
elif opcion == 11:
    simular_cola_revision(historialcursos)
elif opcion == 12:
    mostrarhistorial(historialgeneral)
elif opcion == 13:
    print(f"Gracias por usar el sistema, {nom_pers}, adiosito")
    break
else:
    print("ingrese una opcion valida")
```

descripción de cada fase

inicialización	*registro de notas*

	`n = int(input("ingrese la cantidad de cursos que desea registrar: "))`
	
	al ingresar pide al usuario la cantidad de cursos que desea registrar (n).
	
	`for i in range(n):`
	
	Inicia un bucle que va desde 0 hasta n-1 para registrar n cursos.
	
	`nombre = input(f"ingrese el nombre del curso {i + 1}: ").strip()`
	
	Pide al usuario el nombre del curso .strip() elimina cualquier espacio antes o después del nombre.
	
	`if not nombre:`
	`print("el nombre del curso no puede estar vacío.")`
	`return`
	
	Verifica si el nombre está vacío (es decir, si no se ingresó nada), Si está vacío, imprime un mensaje de error y termina la función con return.
	
	`try:`
	`nota = float(input("ingrese la nota obtenida (0-100): "))`
	
	Pide la nota del curso y la convierte a float. Esto es para permitir notas con decimales
	
	`if nota < 0 or nota > 100:`
	`print("la nota debe estar entre 0 y 100.")`
	`return`
	
	Si la nota no está en el rango entre 0 y 100, muestra un mensaje de error y termina la función con return.
	
	`except ValueError:`
	`print("la nota debe ser un numero.")`
	`return`
	
	Si ocurre un error al intentar convertir la entrada a float (es decir, si el usuario ingresa algo que no es un número), captura la excepción ValueError y muestra un mensaje de error.
	
	`print(f"curso {nombre} registrado correctamente con nota {nota}.")`
	
	Si todo sale bien, muestra un mensaje indicando que el curso y la nota se registraron correctamente.

mostrar cursos	*promedio general*

`def mostrarcursos(historialcursos):`
 	
def: Palabra clave para definir una función.
mostrarcursos: Nombre de la función, que indica que su propósito es mostrar los cursos registrados.
historialcursos: Parámetro que se espera sea un diccionario donde las claves son nombres de cursos y los valores son las notas.
 	
`if not historialcursos:`
`print("no hay cursos registrados.")`
 	
Verifica si el diccionario historialcursos está vacío. **not historialcursos** es una forma de decir: Está vacío o no tiene datos, Si el diccionario está vacío, muestra este mensaje en pantalla.
 	
`else:`
`print("listado de cursos registrados:")`
 	
Si el diccionario **no** está vacío, ejecuta el bloque siguiente y muestra un encabezado indicando que se va a listar los cursos.
 	
`for nombre, nota in historialcursos.items():`
 	
Recorre cada par clave-valor del diccionario. **nombre** es el nombre del curso (la clave). **nota** es la nota asociada a ese curso (el valor) **.items()** devuelve una lista de tuplas con cada curso y su nota.
 	
`print(f"{nombre} → nota: {nota}")`
 	
Imprime cada curso y su nota en un formato entendible. **f"{...}"** es una cadena formateada (f-string) que permite insertar variables directamente.
	`def promediogeneral(historialcursos):`
	
	Define una función llamada promediogeneral. Recibe un parámetro historialcursos, que se espera sea un diccionario con nombres de cursos como claves y notas como valores.
	
	`if not historialcursos:`
	`print("no hay cursos registrados para calcular el promedio.")`
	`return`
	
	Verifica si el diccionario está vacío. **not historialcursos** evalúa a True si no hay cursos registrados. Si no hay cursos, muestra un mensaje y termina la función con return.
	
	`total = sum(historialcursos.values())`
	
	historialcursos.values() devuelve una lista de todas las notas. **sum(...)** suma esos valores.
	
	`promedio = total / len(historialcursos)`
	
	Calcula el promedio dividiendo el total de notas entre la cantidad de cursos. **len(historialcursos)** da el número de cursos registrados.
	
	`print(f"promedio general calculado: {promedio:.2f}")`
	
	Muestra el promedio con dos decimales **{promedio:.2f}** es una f-string que formatea el número con dos cifras después del punto decimal.

cursos aprobados y reprobados

```
def reproapro(historialcursos):
```

Define una función llamada reproapro. Recibe un parámetro historialcursos, que se espera sea un diccionario con nombres de cursos como claves y notas como valores.

```
    aprobados = 0
    reprobados = 0
```

aprobados: cuenta cuántos cursos tienen nota aprobatoria. reprobados: cuenta cuántos cursos tienen nota reprobatoria.

```
    for nota in historialcursos.values():
```

Recorre todas las notas del diccionario **.values()** devuelve solo los valores (las notas), ignorando los nombres de los cursos.

```
        if nota >= 61:
            aprobados += 1
        else:
            reprobados += 1
```

Si la nota es mayor o igual a 61, se considera aprobada y se incrementa el contador aprobados. Si es menor a 61, se incrementa el contador reprobados.

```
    print(f"total de cursos aprobados: {aprobados}")
    print(f"total de cursos reprobados: {reprobados}")
```

Muestra en pantalla cuántos cursos fueron aprobados y cuántos reprobados. Usa **f-strings** para insertar los valores directamente en el texto.

buscar curso de forma lineal

```
def busquedalineal_func(historialcursos, busquedalineal, historialgeneral):
```

Define la función **busquedalineal_func.** Recibe tres parámetros: **historialcursos:** diccionario con cursos y sus notas. **busquedalineal:** diccionario que lleva el conteo de búsquedas por curso. **historialgeneral:** lista que guarda mensajes de acciones realizadas.

```
    nombre = input("ingrese el nombre del curso a buscar: ").strip()
```

Solicita al usuario el nombre del curso a buscar .strip() elimina espacios al inicio y al final del texto ingresado

```
    if not nombre:
        print("este campo no puede estar vacio")
        return
```

Verifica si el usuario no ingresó nada. Si el campo está vacío, muestra un mensaje y termina la función.

```
    encontrado = False
Variable de control para saber si el curso fue encontrado.
    for curso in historialcursos:
```

Recorre cada nombre de curso en el diccionario **historialcursos**.

```
        if curso.lower() == nombre.lower():
```

Compara el nombre ingresado con el nombre del curso, ignorando mayúsculas/minúsculas.

```
            print(f"curso encontrado: {curso} // nota: {historialcursos[curso]}")
```

Muestra el nombre del curso y su nota.

```
            busquedalineal[curso] = busquedalineal.get(curso, 0) + 1
```

Actualiza el contador de búsquedas para ese curso.Si el curso no ha sido buscado antes, lo inicia en 0 y suma 1.

```
            print(f"este curso ha sido consultado {busquedalineal[curso]} veces.")
```

Muestra cuántas veces se ha buscado ese curso.

```
            historialgeneral.append(f"Se busco el curso {curso} - total de busquedas:
{busquedalineal[curso]}")
```

Agrega un mensaje al historial general indicando la búsqueda exitosa y el número de veces que se ha buscado.

```
            encontrado = True
            break
```

Marca que el curso fue encontrado y termina el bucle.

```
    if not encontrado:
        print("curso no encontrado.")
        historialgeneral.append(f"busqueda fallida: curso {nombre} no
encontrado.")
```

Si no se encontró el curso, muestra un mensaje y lo registra en el historial como búsqueda fallida.

actualizar nombre del curso, nota de curso o ambos

```
def editarcursonota(historialcursos, historialgeneral):
```

Define la función **editarcursonota**. Recibe dos parámetros:
historialcursos: diccionario con cursos y sus notas.
historialgeneral: lista para registrar los cambios realizados.

```
nombre = input("ingrese el nombre del curso que desea actualizar: ").strip()
```

Pide al usuario el nombre del curso a editar .strip() elimina espacios al inicio y al final

```
if not nombre:
    print("este campo no puede estar vacío.")
    return
```

Si el usuario no escribe nada, muestra un mensaje y termina la función.

```
if nombre not in historialcursos:
    print(f"el curso '{nombre}' no esta registrado.")
historialgeneral.append(f"intento fallido de actualizacion: curso {nombre} no encontrado.")
return
```

Verifica si el curso existe en el diccionario. Si no existe, informa al usuario y registra el intento fallido en el historial.

```
cursoactual = nombre
notaactual = historialcursos[cursoactual]
```

Guarda el nombre y la nota actual del curso para trabajar con ellos durante la edición.

```
while True:
```

Inicia un bucle que permite editar varias veces hasta que el usuario confirme los cambios.

```
    print(f"curso seleccionado: {cursoactual}  nota actual: {notaactual}")
    print("1. editar nombre del curso")
    print("2. editar nota del curso")
    print("3. confirmar cambios")
```

Muestra el curso actual y las opciones disponibles.

```
    try:
        opcion = int(input("seleccione una opcion: "))
    except ValueError:
        print("ingrese una opción valida.")
        continue
```

Solicita una opción numérica. Si el usuario ingresa algo no numérico, muestra un error y vuelve a pedir.

```
    if opcion == 1:
        nuevonombre = input("ingrese el nuevo nombre del curso: ").strip()
        if not nuevonombre:
            print("este campo no puede estar vacio.")
            continue
```

eliminar curso

```
def eliminarcurso(historialcursos, historialgeneral):
```

Define la función **eliminarcurso**. Recibe dos parámetros:
historialcursos: diccionario con cursos y sus notas.
historialgeneral: lista que registra las acciones realizadas.

```
if not historialcursos:
    print("no hay cursos registrados para eliminar.")
    return
```

Verifica si el diccionario está vacío. Si no hay cursos, muestra un mensaje y termina la función.

```
nombre = input("ingrese el nombre del curso que desea eliminar: ").strip()
```

Solicita al usuario el nombre del curso que quiere eliminar **.strip()** elimina espacios al inicio y al final

```
if not nombre:
    print("este campo no puede estar vacio.")
    return
```

Si el usuario no escribe nada, muestra un mensaje y termina la función.

```
curso_encontrado = None
```

Inicializa una variable para guardar el nombre del curso si se encuentra.

```
for curso in historialcursos:
    if curso.lower() == nombre.lower():
        curso_encontrado = curso
        break
```

Recorre los cursos registrados. Compara el nombre ingresado con cada curso, ignorando mayúsculas/minúsculas. Si lo encuentra, guarda el nombre original y sale del bucle.

```
if curso_encontrado:
```

Verifica si se encontró el curso

```
confirmacion = input(f"seguro que desea eliminar {curso_encontrado} (s/n): ").lower()
    if confirmacion == 's':
        historialcursos.pop(curso_encontrado)
historialgeneral.append(f"curso eliminado: {curso_encontrado}")
print(f"Curso '{curso_encontrado}' eliminado correctamente")
```

Si el usuario confirma con “s”, elimina el curso del diccionario. Registra la eliminación en el historial. Muestra un mensaje de éxito.

```
    else:
        print("la eliminacion fue cancelada")
```

Si el usuario no confirma, cancela la operación

```
else:
    print(f"el curso {nombre} no fue encontrado.")
historialgeneral.append(f"intento fallido de eliminación: curso {nombre} no existe.")
```

```
        if nuevonombre in historialcursos:
            print("ya existe un curso con ese nombre.")
            continue
        historialcursos[nuevonombre] = historialcursos.pop(cursoactual)
        historialgeneral.append(f"nombre actualizado: '{cursoactual}' → '{nuevonombre}'")
        cursoactual = nuevonombre
        print(f"nombre cambiado a {nuevonombre}.")
```

Pide el nuevo nombre del curso. Verifica que no esté vacío ni duplicado. Actualiza el nombre en el diccionario y en el historial.

```
    elif opcion == 2:
        try:
            nuevanota = float(input("ingrese la nueva nota (0-100): "))
            if nuevanota < 0 or nuevanota > 100:
                print("la nota debe estar entre 0 y 100.")
                continue
        except ValueError:
            print("la nota debe ser un numero.")
            continue
        historialcursos[cursoactual] = nuevanota
        notaactual = nuevanota
        historialgeneral.append(f"nota actualizada para {cursoactual}: {nuevanota}")
        print(f"nota cambiada a {nuevanota}.")
```

Pide una nueva nota. Verifica que sea un número entre 0 y 100. Actualiza la nota en el diccionario y en el historial.

```
    elif opcion == 3:
        print("actualizado... regresando al menu principal...")
        break
```

Sale del bucle y regresa al menú principal.

```
    else:
        print("opción no valida.")
```

Si el número ingresado no es 1, 2 o 3, muestra un mensaje de error.

Si no se encontró el curso, informa al usuario y registra el intento fallido en el historial.

ordenar en base a la nota

```
def ordenar_por_nota(cursos, historial):
    if not cursos:
        print("No hay cursos para ordenar.")
        return
    print("1. De menor a mayor")
    print("2. De mayor a menor")
    opcion = int(input("Elige una opción: "))
    lista = list(cursos.items())  # Convertir el diccionario a lista de tuplas
    for i in range(len(lista)):
        for j in range(len(lista) - 1):
            if opcion == 1 and lista[j][1] > lista[j + 1][1]:
                lista[j], lista[j + 1] = lista[j + 1], lista[j]
            elif opcion == 2 and lista[j][1] < lista[j + 1][1]:
                lista[j], lista[j + 1] = lista[j + 1], lista[j]
    print("Cursos ordenados:")
    for nombre, nota in lista:
        print(f"{nombre} - Nota: {nota}")
        historial.append(f"{nombre} - Nota: {nota}")
```

ordenar en base al nombre

```
def ordenar_por_nombre(cursos, historial):
    if not cursos:
        print("No hay cursos registrados para ordenar.")
        return
    print("¿Cómo deseas ordenar los nombres?")
    print("1. De A a Z")
    print("2. De Z a A")
    try:
        opcion = int(input("Seleccione una opción: "))
    except ValueError:
        print("Opción inválida.")
        return
    lista = list(cursos.items())
    # Ordenamiento por inserción
    for i in range(1, len(lista)):
        clave = lista[i]
        j = i - 1
        # Comparación de nombres (ignorando mayúsculas/minúsculas)
        while j >= 0 and (
            (opcion == 1 and clave[0].lower() < lista[j][0].lower()) or
            (opcion == 2 and clave[0].lower() > lista[j][0].lower())
        ):
            lista[j + 1] = lista[j]
            j -= 1
        lista[j + 1] = clave
    if opcion == 1:
        historial.append("Cursos ordenados por nombre (A-Z) con
inserción")
    elif opcion == 2:
        historial.append("Cursos ordenados por nombre (Z-A) con
inserción")
    print("Cursos ordenados por nombre:")
    for nombre, nota in lista:
        print(f" - {nombre} / Nota: {nota}")
        historial.append(f"Curso: {nombre} / Nota: {nota}")
```

búsqueda de curso en base al nombre de forma binaria	*simulacion*

historial de cambios y búsqueda	*salir*

codigo completo

```
historialcursos = {}
historialgeneral = []
busquedalineal = {}

# 1. Registro de notas de cursos
def registronotacurso():
    n = int(input("ingrese la cantidad de cursos que desea registrar: "))
    for i in range(n):
        nombre = input(f"ingrese el nombre del curso {i + 1}: ").strip()
        if not nombre:
            print("el nombre del curso no puede estar vacío.")
            return
        try:
            nota = float(input("ingrese la nota obtenida (0-100): "))
            if nota < 0 or nota > 100:
                print("la nota debe estar entre 0 y 100.")
                return
        except ValueError:
            print("la nota debe ser un numero.")
            return

        historialcursos[nombre] = nota
        print(f"curso {nombre} registrado correctamente con nota {nota}.")

# 2. Mostrar cursos
def mostrarcursos(historialcursos):
    if not historialcursos:
        print("no hay cursos registrados.")
    else:
        print("listado de cursos registrados:")
        for nombre, nota in historialcursos.items():
            print(f"{nombre} → nota: {nota}")

# 3. Promedio general
def promediogeneral(historialcursos):
    if not historialcursos:
        print("no hay cursos registrados para calcular el promedio.")
        return

    total = sum(historialcursos.values())
    promedio = total / len(historialcursos)
    print(f"promedio general calculado: {promedio:.2f}")
```

```
# 4. Conteo de cursos aprobados y reprobados
def reproapro(historialcursos):
    aprobados = 0
    reprobados = 0
    for nota in historialcursos.values():
        if nota >= 60:
            aprobados += 1
        else:
            reprobados += 1
    print(f"total de cursos aprobados: {aprobados}")
    print(f"total de cursos reprobados: {reprobados}")

# 5. Búsqueda lineal
def busquedalineal_func(historialcursos, busquedalineal, historialgeneral):
    nombre = input("ingrese el nombre del curso a buscar: ").strip()
    if not nombre:
        print("este campo no puede estar vacio")
        return

    encontrado = False
    for curso in historialcursos:
        if curso.lower() == nombre.lower():
            print(f"curso encontrado: {curso} // nota: {historialcursos[curso]}")
            busquedalineal[curso] = busquedalineal.get(curso, 0) + 1
            print(f"este curso ha sido consultado {busquedalineal[curso]} veces.")
            historialgeneral.append(f"Se busco el curso {curso} - total de busquedas: {busquedalineal[curso]}")
            encontrado = True
            break
    if not encontrado:
        print("curso no encontrado.")
        historialgeneral.append(f"busqueda fallida: curso {nombre} no encontrado.")

# 6. Actualizar o editar nombre del curso/nota
def editarcursonota(historialcursos, historialgeneral):
    nombre = input("ingrese el nombre del curso que desea actualizar: ").strip()
    if not nombre:
        print("este campo no puede estar vacío.")
        return
    if nombre not in historialcursos:
        print(f"el curso '{nombre}' no esta registrado.")
        historialgeneral.append(f"intento fallido de actualizacion: curso {nombre} no encontrado.")
```

```
    return
cursoactual = nombre
notaactual = historialcursos[cursoactual]
while True:
    print(f"curso seleccionado: {cursoactual}  nota actual: {notaactual}")
    print("1. editar nombre del curso")
    print("2. editar nota del curso")
    print("3. confirmar cambios")
    try:
        opcion = int(input("seleccione una opcion: "))
    except ValueError:
        print("ingrese una opción valida.")
        continue
    if opcion == 1:
        nuevonombre = input("ingrese el nuevo nombre del curso: ").strip()
        if not nuevonombre:
            print("este campo no puede estar vacio.")
            continue
        if nuevonombre in historialcursos:
            print("ya existe un curso con ese nombre.")
            continue
        historialcursos[nuevonombre] = historialcursos.pop(cursoactual)
        historialgeneral.append(f"nombre actualizado: '{cursoactual}' → '{nuevonombre}'")
        cursoactual = nuevonombre
        print(f"nombre cambiado a {nuevonombre}.")
    elif opcion == 2:
        try:
            nuevanota = float(input("ingrese la nueva nota (0-100): "))
            if nuevanota < 0 or nuevanota > 100:
                print("la nota debe estar entre 0 y 100.")
                continue
        except ValueError:
            print("la nota debe ser un numero.")
            continue
        historialcursos[cursoactual] = nuevanota
        notaactual = nuevanota
        historialgeneral.append(f"nota actualizada para {cursoactual}: {nuevanota}")
        print(f"nota cambiada a {nuevanota}.")
    elif opcion == 3:
        print("actualizado... regresando al menu principal...")
        break
    else:
```

```
            print("opción no valida.")


# 7. Eliminar curso
def eliminarcurso(historialcursos, historialgeneral):
    if not historialcursos:
        print("no hay cursos registrados para eliminar.")
        return
    nombre = input("ingrese el nombre del curso que desea eliminar: ").strip()
    if not nombre:
        print("este campo no puede estar vacio.")
        return
    curso_encontrado = None
    for curso in historialcursos:
        if curso.lower() == nombre.lower():
            curso_encontrado = curso
            break
    if curso_encontrado:
        confirmacion = input(f"seguro que desea eliminar {curso_encontrado} (s/n): ").lower()
        if confirmacion == 's':
            historialcursos.pop(curso_encontrado)
            historialgeneral.append(f"curso eliminado: {curso_encontrado}")
            print(f"Curso '{curso_encontrado}' eliminado correctamente")
        else:
            print("la eliminacion fue cancelada")
    else:
        print(f"el curso {nombre} no fue encontrado.")
        historialgeneral.append(f"intento fallido de eliminación: curso {nombre} no existe.")


# 8. Ordenar por nota (burbuja)
def ordenar_por_nota(cursos, historial):
    if not cursos:
        print("No hay cursos para ordenar.")
        return

    print("I. De menor a mayor")
    print("2. De mayor a menor")
    opcion = int(input("Elige una opción: "))

    lista = list(cursos.items())  # Convertir el diccionario a lista de tuplas

    # Algoritmo burbuja clásico
    for i in range(len(lista)):
```

```
        for j in range(len(lista) - 1):
            if opcion == 1 and lista[j][1] > lista[j + 1][1]:
                lista[j], lista[j + 1] = lista[j + 1], lista[j]
            elif opcion == 2 and lista[j][1] < lista[j + 1][1]:
                lista[j], lista[j + 1] = lista[j + 1], lista[j]

    print("Cursos ordenados:")
    for nombre, nota in lista:
        print(f"{nombre} - Nota: {nota}")
        historial.append(f"{nombre} - Nota: {nota}")

# 9. Ordenar por nombre (inserción)
def ordenar_por_nombre(cursos, historial):
    if not cursos:
        print("No hay cursos registrados para ordenar.")
        return
    print("¿Cómo deseas ordenar los nombres?")
    print("1. De A a Z")
    print("2. De Z a A")
    try:
        opcion = int(input("Seleccione una opción: "))
    except ValueError:
        print("Opción inválida.")
        return

    lista = list(cursos.items())

    # Ordenamiento por inserción
    for i in range(1, len(lista)):
        clave = lista[i]
        j = i - 1

        # Comparación de nombres (ignorando mayúsculas/minúsculas)
        while j >= 0 and (
            (opcion == 1 and clave[0].lower() < lista[j][0].lower()) or
            (opcion == 2 and clave[0].lower() > lista[j][0].lower())
        ):
            lista[j + 1] = lista[j]
            j -= 1
        lista[j + 1] = clave

    if opcion == 1:
```

```
        historial.append("Cursos ordenados por nombre (A-Z) con inserción")
    elif opcion == 2:
        historial.append("Cursos ordenados por nombre (Z-A) con inserción")

    print("Cursos ordenados por nombre:")
    for nombre, nota in lista:
        print(f" - {nombre} / Nota: {nota}")
        historial.append(f"Curso: {nombre} / Nota: {nota}")

# 10. Búsqueda binaria
def busqueda_binaria(cursos, historial):
    if not cursos:
        print("No hay cursos registrados.")
        return

    nombre = input("Ingrese el nombre del curso a buscar (binaria): ").strip()
    if not nombre:
        print("El nombre no puede estar vacío.")
        return

    nombres_ordenados = sorted(cursos.keys(), key=lambda x: x.lower())
    izquierda = 0
    derecha = len(nombres_ordenados) - 1
    encontrado = False

    while izquierda <= derecha:
        medio = (izquierda + derecha) // 2
        curso_medio = nombres_ordenados[medio]

        if curso_medio.lower() == nombre.lower():
            nota = cursos[curso_medio]
            print(f"Curso encontrado: {curso_medio} / Nota: {nota}")
            historial.append(f"Búsqueda binaria exitosa: '{curso_medio}' con nota {nota}")
            encontrado = True
            break
        elif nombre.lower() < curso_medio.lower():
            derecha = medio - 1
        else:
            izquierda = medio + 1

    if not encontrado:
        print("Curso no encontrado.")
```

```
        historial.append(f"Búsqueda binaria fallida: curso '{nombre}' no existe.")

# I1. Simular cola de revisión
def simular_cola_revision(historialcursos):
    if not historialcursos:
        print("No hay cursos en el historial.")
        return

    nom2 = input("confirme su nombre para iniciar la revisión: ")
    if nom2.strip() == "":
        print("El nombre no puede estar vacío.")
        return
    else:
        nom2 == nom_pers.strip()
    if nom2 != nom_pers:
        print("El nombre no coincide con el registrado. Revisión cancelada.")
        return
    for curso, nota in historialcursos.items():
        print(f"Revisando curso: {curso} / Nota: {nota}")

    print(f"Hola {nom2}, comenzando la revisión de cursos...")
    print("Simulando cola de revisión...")
    print("Revisión de todos los cursos completada.")
    print(f"Gracias por tu paciencia, {nom2}. Revisión finalizada.")
    print("regresando al menu principal...")

# I2. Mostrar historial de cambios
def mostrarhistorial(historialgeneral):
    if not historialgeneral:
        print("no hay historial de acciones.")
    else:
        print("historial de acciones realizadas:")
        for evento in historialgeneral:
            print(f"- {evento}")

# Menú principal
print("bienvenido al sistema de registro de notas")
nom_pers = input("Ingrese su nombre para empezar: ")
if nom_pers == "":
    print("El nombre no puede estar vacío. Reinicie el programa.")
    exit()
```

```
print("Nombre registrado, ¡HERE WE GO!")


while True:
    print("Menú principal:")
    print("1. Agregar curso")
    print("2. Mostrar cursos")
    print("3. Calcular promedio")
    print("4. Conteo de cursos aprobados y reprobados")
    print("5. Buscar curso por nombre (lineal)")
    print("6. Actualizar o editar nombre del curso/nota")
    print("7. Eliminar curso")
    print("8. Ordenar por nota")
    print("9. Ordenar por nombre")
    print("10. Buscar curso por nombre (binaria)")
    print("11. Simular cola de revisión")
    print("12. Mostrar historial de cambios")
    print("13. Salir")

    try:
        opcion = int(input("Ingrese el número de la opción: "))
    except ValueError:
        print("Por favor, ingrese un número válido.")
        continue

    if opcion == 1:
        registronotacurso()
    elif opcion == 2:
        mostrarcursos(historialcursos)
    elif opcion == 3:
        promediogeneral(historialcursos)
    elif opcion == 4:
        reproapro(historialcursos)
    elif opcion == 5:
        busquedalineal_func(historialcursos, busquedalineal, historialgeneral)
    elif opcion == 6:
        editarcursonota(historialcursos, historialgeneral)
    elif opcion == 7:
        eliminarcurso(historialcursos, historialgeneral)
    elif opcion == 8:
        ordenar_por_nota(cursos=historialcursos, historial=historialgeneral)
    elif opcion == 9:
        ordenar_por_nombre(cursos=historialcursos, historial=historialgeneral)
```

```
elif opcion == 10:
    busqueda_binaria(cursos=historialcursos, historial=historialgeneral)
elif opcion == 11:
    simular_cola_revision(historialcursos)
elif opcion == 12:
    mostrarhistorial(historialgeneral)
elif opcion == 13:
    print(f"Gracias por usar el sistema, {nom_pers}, adiosito")
    break
else:
    print("ingrese una opcion valida")
```

Conclusión

si bien existen formas mas practicas para realizar dichas funciones el poder practicar de esta manera hace que el conocimiento se expanda un poco mas

al realizar dichas funciones ayuda a poder estar con una mente abierta y implementar cosas diferentes sin cambiar el concepto centrar del proyecto, implemente algunas adiciones a las que pedía el proyecto central pero esto no afecto el funcionamiento. para poder obtener este resultado se realizaron varias horas de dedicación y horas de prueba y error, sin embargo, fue difícil el poder llegar este resultado.