**UNIVERSIDAD PRIVADA DOMINGO SAVIO**



**Ejercicios de programación II: Actividad 02**

**02/07/2025 7:49:50**

**Docente:**

Jimmy Nataniel Requena Llorentty

**Estudiante:**

Rebeca Vargas Orellana

**Materia:**

Programación II – Turno mañana

*Santa Cruz – Bolivia*

Ejercicios de Programación

Actividad 02: Clases de Programación II

|  |
| --- |
| def ordenamiento\_burbuja(lista):  n = len(lista)  for i in range (n - 1):  hubo\_intercambio = False  for j in range (n - 1 - i):  if lista[j] > lista[j + 1]:  lista[j], lista [j +1] = lista [j + 1], lista[j]  hubo\_intercambio = True  if not hubo\_intercambio:  break  return lista  if \_\_name\_\_ == "\_\_main\_\_":  numeros = [6, 3, 8, 2, 5]  print("Antes:", numeros)  ordenamiento\_burbuja(numeros)  print("Despues:", numeros) |
|  |
| Código: Ordenamiento burbuja |

|  |
| --- |
| # Definición de la función de ordenamiento burbuja  def ordenamiento\_burbuja(lista):  n = len(lista)  for i in range(n):  for j in range(0, n - i - 1):  if lista[j] > lista[j + 1]:  lista[j], lista[j + 1] = lista[j + 1], lista[j]  #llamar a la función:  lista\_a\_ordenar = [64, 34, 25, 12, 22, 11, 90]  print(f"Lista original: {lista\_a\_ordenar}")  ordenamiento\_burbuja(lista\_a\_ordenar) # Llamamos a la función  print(f"Lista ordenada: {lista\_a\_ordenar}")  # Prueba con assert:  # Caso 1: Lista desordenada  lista1 = [6, 3, 8, 2, 5]  ordenamiento\_burbuja(lista1)  assert lista1 == [2, 3, 5, 6, 8], "Falló en Caso 1"  # Caso 2: Lista ya ordenada  lista2 = [1, 2, 3, 4, 5]  ordenamiento\_burbuja(lista2)  assert lista2 == [1, 2, 3, 4, 5], "Falló en Caso 2"  # Caso 3: Lista ordenada a la inversa (peor caso)  lista3 = [5, 4, 3, 2, 1]  ordenamiento\_burbuja(lista3)  assert lista3 == [1, 2, 3, 4, 5], "Falló en Caso 3"  # Caso 4: Lista con elementos duplicados  lista4 = [5, 1, 4, 2, 5, 5, 2]  ordenamiento\_burbuja(lista4)  assert lista4 == [1, 2, 2, 4, 5, 5, 5], "Falló en Caso 4"  # Casos borde  assert ordenamiento\_burbuja([]) == None  assert ordenamiento\_burbuja([42]) == None, "Fallo en caso borde"  print("¡Todas las pruebas pasaron!") |
|  |
| Código: Ordenamiento de menor a mayor |

|  |
| --- |
|  |
| matriz= [1, 2, 3], [4, 5, 6], [7, 8, 9]  # OPCION 1: Recorriendo con indices  num\_filas = len(matriz)  num\_columnas = len(matriz[0])  for i in range(num\_filas):  for j in range(num\_columnas):  elemento = matriz[i][j]  print(f"elemento en ({i},{j} es {elemento}")  # OPCION 2: Recorriendo por elemento (mas "pythonico")  for fila\_actual in matriz:  for elemento in fila\_actual:  print(elemento, end="")  print() |
|  |
| Código:RecorriendoConIndices |

|  |
| --- |
| teclado= [  ["[0]", "[2]", "[3]"],  ["[4]", "[5]", "[6]"],  ["[7]", "[8]", "[9]"]  ]  # OPCION 1: Recorriendo con indices  num\_filas = len(teclado)  num\_columnas = len(teclado[0])  for i in range(num\_filas):  for j in range(num\_columnas):  tecla=teclado[i][j]  print(f"Tecla en ({i},{j} es {tecla}")  # OPCION 2: Recorriendo por elemento (mas "pythonico")  for fila in teclado:  for tecla in fila:  print(tecla, end="")  print() |
|  |
| Código:Matriz\_RecorriendoConIndices |

|  |
| --- |
| matriz= [0, 2, 3], [4, 5, 6], [7, 8, 9]  # OPCION 1: Recorriendo con indices  num\_filas = len(matriz)  num\_columnas = len(matriz[0])  for i in range(num\_filas):  for j in range(num\_columnas):  elemento = matriz[i][j]  print(f"elemento en ({i},{j} es {elemento}")  # OPCION 2: Recorriendo por elemento (mas "pythonico")  for fila\_actual in matriz:  for elemento in fila\_actual:  print(elemento, end="\t")  print() |
|  |
| Codigo:matriz\_telefono |

|  |
| --- |
| # Definimos la función que suma los elementos de la diagonal principal de una matriz cuadrada  def sumar\_diagonal\_principal(matriz):  """  Esta función recibe una matriz cuadrada (misma cantidad de filas y columnas)  y retorna la suma de los elementos en su diagonal principal.  Ejemplo:  matriz = [[1, 2],  [3, 4]]  diagonal principal: 1 y 4 → suma = 5  """  suma = 0  for i in range(len(matriz)):  suma += matriz[i][i] # Accede al elemento en la posición (i, i)  return suma  # Función de prueba para verificar que sumar\_diagonal\_principal funciona correctamente  def probar\_suma\_diagonal\_principal():  print("\nProbando sumar\_diagonal\_principal...")  # Caso 1: matriz 3x3 con números consecutivos  m1 = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]  assert sumar\_diagonal\_principal(m1) == 15 # 1 + 5 + 9  # Caso 2: matriz 2x2 con ceros y valores definidos  m2 = [[10, 0], [0, 20]]  assert sumar\_diagonal\_principal(m2) == 30 # 10 + 20  # Caso borde: matriz 1x1  m3 = [[5]]  assert sumar\_diagonal\_principal(m3) == 5 # Solo un elemento en la diagonal  print("¡Pruebas para sumar\_diagonal\_principal pasaron! ")  # Llamamos a la función para ejecutar las pruebas  probar\_suma\_diagonal\_principal() |
|  |
| Codigo: Suma diagonal |

|  |
| --- |
| # Definimos la función que suma los elementos por cada fila de la matriz  def sumar\_por\_filas(matriz):  """  Esta función recibe una matriz (lista de listas)  y devuelve una lista con la suma de cada fila.  Ejemplo:  matriz = [[1, 2, 3], [4, 5, 6]]  resultado = [6, 15]  """  resultado = []  for fila in matriz:  suma\_fila = sum(fila) # Suma todos los elementos de la fila  resultado.append(suma\_fila)  return resultado  # Función de prueba para verificar que sumar\_por\_filas funciona correctamente  def probar\_suma\_por\_filas():  print("\nProbando sumar\_por\_filas...")  # Caso 1: matriz con 3 filas y 3 columnas  m1 = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]  assert sumar\_por\_filas(m1) == [6, 15, 24] # 1+2+3, 4+5+6, 7+8+9  # Caso 2: matriz con pares repetidos  m2 = [[10, 10], [20, 20], [30, 30]]  assert sumar\_por\_filas(m2) == [20, 40, 60]  # Caso borde: matriz vacía  assert sumar\_por\_filas([]) == [] # No hay filas que sumar  print("¡Pruebas para sumar\_por\_filas pasaron! ")  # Llamamos a la función para ejecutar las pruebas  probar\_suma\_por\_filas() |
|  |
| Codigo: Suma por filas |

|  |
| --- |
| # Definimos la función que suma todos los elementos de una matriz  def sumar\_total\_matriz(matriz):  """  Esta función recibe una matriz (lista de listas)  y retorna la suma total de todos sus elementos.  Ejemplo:  matriz = [[1, 2], [3, 4]]  resultado = 10  """  total = 0  for fila in matriz:  for elemento in fila:  total += elemento  return total  # Función para probar que sumar\_total\_matriz funciona correctamente  def probar\_suma\_total():  print("Probando sumar\_total\_matriz...")  # Caso 1: matriz normal  m1 = [[1, 2, 3], [4, 5, 6]]  assert sumar\_total\_matriz(m1) == 21 # 1+2+3+4+5+6 = 21  # Caso 2: matriz con negativos y ceros  m2 = [[-1, 0, 1], [10, -5, 5]]  assert sumar\_total\_matriz(m2) == 10 # -1+0+1+10-5+5 = 10  # Casos borde o límites  assert sumar\_total\_matriz([[]]) == 0 # Matriz con una fila vacía  assert sumar\_total\_matriz([]) == 0 # Matriz completamente vacía  assert sumar\_total\_matriz([[42]]) == 42 # Matriz de un solo elemento  print("¡Pruebas para sumar\_total\_matriz pasaron! ")  # Llamamos a la función de pruebas  probar\_suma\_total() |
|  |
| Codigo: Analisis Matricial |

|  |
| --- |
| def transponer\_matriz(matriz):  if not matriz or not matriz[0]:  return []  num\_filas = len(matriz)  num\_columnas = len(matriz[0])  matriz\_transpuesta = []  for j in range(num\_columnas):  nueva\_fila = []  for i in range(num\_filas):  nueva\_fila.append(matriz[i][j])  matriz\_transpuesta.append(nueva\_fila)  return matriz\_transpuesta  m1 = [[1, 2, 3], [4, 5, 6]]  t1 = transponer\_matriz(m1)  assert t1 == [[1, 4], [2, 5], [3, 6]]  print("Prueba 1 (2x3) pasada!") |
|  |
| Código: tansponer una matriz |

|  |
| --- |
| producto = {'codigo': 'P001', 'nombre': 'Café', 'precio': 38.0, 'stock': 100}  print("\n--- Claves del producto ---")  for clave in producto: # Por defecto, itera sobre las CLAVES  print(clave)  print("\n--- Clave y Valor ---")  for clave in producto:  valor = producto[clave]  print(f"{clave.capitalize()}: {valor}") |
|  |
| Codigo:Diccionario bucle for |

|  |
| --- |
| #Paso 1: Variables Globales  lista\_de\_tareas = []  proximo\_id\_tarea = 1 # Para generar IDs únicos  #Paso 2: Implementar agregar\_tarea  def agregar\_tarea(descripcion, prioridad="media"):  global proximo\_id\_tarea # ¡Necesario para modificar una variable global!  nueva\_tarea = {  "id": proximo\_id\_tarea,  "descripcion": descripcion,  "completada": False,  "prioridad": prioridad  }  lista\_de\_tareas.append(nueva\_tarea)  proximo\_id\_tarea += 1  print(f" Tarea '{descripcion}' añadida con éxito.")  #Paso 3: Implementar mostrar\_tareas  def mostrar\_tareas():  print("\n--- LISTA DE TAREAS ---")  if not lista\_de\_tareas:  print("¡No hay tareas pendientes! ¡A disfrutar!")  return  for tarea in lista\_de\_tareas:  estado = " " if tarea["completada"] else " "  print(f"{estado} ID: {tarea['id']} | {tarea['descripcion']} (Prioridad: {tarea['prioridad']})")  print("------------------------")  #Prueba tus funciones:  agregar\_tarea("Estudiar para el examen de Cálculo")  agregar\_tarea("Hacer las compras", prioridad="alta")  mostrar\_tareas() |
|  |
| Codigo: Buscar Tarea por ID |

|  |
| --- |
| #Paso 1: Variables Globales  lista\_de\_tareas = []  proximo\_id\_tarea = 1 # Para generar IDs únicos  #Paso 2: Implementar agregar\_tarea  def agregar\_tarea(descripcion, prioridad="media"):  global proximo\_id\_tarea # ¡Necesario para modificar una variable global!  nueva\_tarea = {  "id": proximo\_id\_tarea,  "descripcion": descripcion,  "completada": False,  "prioridad": prioridad  }  lista\_de\_tareas.append(nueva\_tarea)  proximo\_id\_tarea += 1  print(f" Tarea '{descripcion}' añadida con éxito.")  #Paso 3: Implementar mostrar\_tareas  def mostrar\_tareas():  print("\n--- LISTA DE TAREAS ---")  if not lista\_de\_tareas:  print("¡No hay tareas pendientes! ¡A disfrutar!")  return  for tarea in lista\_de\_tareas:  estado = " " if tarea["completada"] else " "  print(f"{estado} ID: {tarea['id']} | {tarea['descripcion']} (Prioridad: {tarea['prioridad']})")  print("------------------------")  #Prueba tus funciones:  agregar\_tarea("Estudiar para el examen de Cálculo")  agregar\_tarea("Hacer las compras", prioridad="alta")  mostrar\_tareas()  #Paso 4: Implementar buscar\_tarea\_por\_id  def buscar\_tarea\_por\_id(id\_buscado):  """Recorre la lista de tareas y devuelve el diccionario de la tarea  que coincide con el id\_buscado. Si no la encuentra, devuelve None."""  for tarea in lista\_de\_tareas:  if tarea["id"] == id\_buscado:  return tarea # ¡Éxito! Devolvemos el diccionario completo  return None # Si el bucle termina, no se encontró  #Prueba tu función de búsqueda:  # Asumiendo que ya agregaste tareas con IDs 1 y 2...  tarea\_encontrada = buscar\_tarea\_por\_id(1)  if tarea\_encontrada:  print(f"\nBúsqueda exitosa: {tarea\_encontrada['descripcion']}")  else:  print("\nBúsqueda fallida: Tarea no encontrada.")  tarea\_fantasma = buscar\_tarea\_por\_id(99)  if not tarea\_fantasma:  print("Búsqueda de tarea inexistente funcionó correctamente.")  #Paso 5: Implementar marcar\_tarea\_completada  def marcar\_tarea\_completada(id\_tarea):  # ¡Reutilizamos nuestra función de búsqueda!  tarea = buscar\_tarea\_por\_id(id\_tarea)  if tarea: # Si la búsqueda devolvió un diccionario (no None)  tarea["completada"] = True  print(f" Tarea '{tarea['descripcion']}' marcada como completada.")  else:  print(f" Error: No se encontró la tarea con ID {id\_tarea}.")  #Paso 6: Implementar eliminar\_tarea  def eliminar\_tarea(id\_tarea):  tarea = buscar\_tarea\_por\_id(id\_tarea)  if tarea:  lista\_de\_tareas.remove(tarea)  print(f" Tarea '{tarea['descripcion']}' eliminada.")  else:  print(f" Error: No se encontró la tarea con ID {id\_tarea}.")  #Prueba todo el flujo:  mostrar\_tareas()  marcar\_tarea\_completada(1)  mostrar\_tareas() # Debería mostrar la tarea 1 como completada  eliminar\_tarea(2)  mostrar\_tareas() # La tarea 2 ya no debería aparecer  marcar\_tarea\_completada(99) # Probar con un ID que no existe |
|  |
| código: buscar tareas por ID |

|  |
| --- |
| class Libro:  """  Clase que representa un libro con sus atributos principales.  """  def \_\_init\_\_(self, titulo, autor, isbn, paginas):  """  Constructor de la clase Libro.  Args:  titulo (str): Título del libro  autor (str): Autor del libro  isbn (str): ISBN del libro  paginas (int): Número de páginas del libro  """  # Crear los atributos de instancia correspondientes  self.titulo = titulo  self.autor = autor  self.isbn = isbn  self.paginas = paginas  # Atributo extra que se inicializa siempre por defecto  self.disponible = True  def mostrar\_info(self):  """  Método que imprime todos los atributos del libro de forma clara y formateada.  """  print("=" \* 50)  print("INFORMACIÓN DEL LIBRO")  print("=" \* 50)  print(f"Título: {self.titulo}")  print(f"Autor: {self.autor}")  print(f"ISBN: {self.isbn}")  print(f"Páginas: {self.paginas}")  print(f"Disponible: {'Sí' if self.disponible else 'No'}")  print("=" \* 50)  # Ejemplo de uso (no te preocupes por crear objetos todavía, ¡solo define la clase!)  if \_\_name\_\_ == "\_\_main\_\_":  # Creación de ejemplo para demostrar el funcionamiento  libro1 = Libro("Cien años de soledad", "Gabriel García Márquez", "978-0307474728", 417)  libro1.mostrar\_info()  # Cambiar disponibilidad  libro1.disponible = False  print("\nDespués de cambiar disponibilidad:")  libro1.mostrar\_info() |
|  |
| código: poo\_ej1 |

|  |
| --- |
| # Ejercicio 2: Creación de instancias y métodos de comportamiento  class Libro:  """  Clase que representa un libro con sus atributos principales.  """  def \_init\_(self, titulo, autor, isbn, paginas):  """  Constructor de la clase Libro.  """  self.titulo = titulo  self.autor = autor  self.isbn = isbn  self.paginas = paginas  self.disponible = True  def mostrar\_info(self):  """  Método que imprime todos los atributos del libro de forma clara y formateada.  """  print("=" \* 50)  print("INFORMACIÓN DEL LIBRO")  print("=" \* 50)  print(f"Título: {self.titulo}")  print(f"Autor: {self.autor}")  print(f"ISBN: {self.isbn}")  print(f"Páginas: {self.paginas}")  print(f"Disponible: {'Sí' if self.disponible else 'No'}")  print("=" \* 50)  def prestar\_libro(self):  """  Método para prestar el libro. Cambia disponible a False si está disponible.  """  if self.disponible == True:  self.disponible = False  print(f"El libro '{self.titulo}' ha sido prestado.")  else:  print(f"El libro '{self.titulo}' ya está prestado.")  def devolver\_libro(self):  """  Método para devolver el libro. Cambia disponible a True si estaba prestado.  """  if self.disponible == False:  self.disponible = True  print(f"El libro '{self.titulo}' ha sido devuelto.")  else:  print(f"El libro '{self.titulo}' ya estaba disponible.")  # 1. Definición de clase Libro (arriba) ✓  # 2. Crear dos objetos Libro diferentes  libro1 = Libro("El Principito", "Antoine de Saint-Exupéry", "978-3-14-046401-7", 120)  libro2 = Libro("Raza de Bronce", "Alcides Arguedas", "978-99905-2-213-9", 250)  # 3. Acceder y mostrar algunos de sus atributos directamente  print(f"\nEl autor del primer libro es: {libro1.autor}")  print(f"El ISBN del segundo libro es: {libro2.isbn}")  # 4. Llamar al método mostrar\_info() para cada uno de los objetos  print("\n--- Mostrando información completa ---")  libro1.mostrar\_info()  libro2.mostrar\_info()  # 5. Añadir métodos de comportamiento (ya implementados arriba) ✓  # 6. Prueba los nuevos métodos con tus objetos libro1 y libro2  print("\n--- Probando métodos de comportamiento ---")  # Prestar libro1  libro1.prestar\_libro()  # Intentar prestar libro1 otra vez (ya prestado)  libro1.prestar\_libro()  # Devolver libro1  libro1.devolver\_libro()  # Intentar devolver libro1 otra vez (ya disponible)  libro1.devolver\_libro()  print("\n--- Probando con libro2 ---")  # Prestar libro2  libro2.prestar\_libro()  # Devolver libro2  libro2.devolver\_libro()  print("\n--- Estado final de los libros ---")  libro1.mostrar\_info()  libro2.mostrar\_info() |
|  |
| código: poo\_ej2 |

|  |
| --- |
| class Libro:  """  Clase que representa un libro con sus atributos principales.  """  def \_\_init\_\_(self, titulo, autor, isbn, paginas):  """  Constructor de la clase Libro.  Args:  titulo (str): Título del libro  autor (str): Autor del libro  isbn (str): ISBN del libro  paginas (int): Número de páginas del libro  """  self.titulo = titulo  self.autor = autor  self.isbn = isbn  self.paginas = paginas  self.disponible = True  def mostrar\_info(self):  """  Método que imprime todos los atributos del libro de forma clara y formateada.  """  print("=" \* 50)  print("INFORMACIÓN DEL LIBRO")  print("=" \* 50)  print(f"Título: {self.titulo}")  print(f"Autor: {self.autor}")  print(f"ISBN: {self.isbn}")  print(f"Páginas: {self.paginas}")  print(f"Disponible: {'Sí' if self.disponible else 'No'}")  print("=" \* 50)  if \_\_name\_\_ == "\_\_main\_\_":  # Crear objetos de tipo Libro  libro1 = Libro("Cien años de soledad", "Gabriel García Márquez", "978-0307474728", 417)  libro2 = Libro("1984", "George Orwell", "978-0451524935", 328)  libro3 = Libro("El Principito", "Antoine de Saint-Exupéry", "978-0156013987", 96)  # Crear una lista vacía  mi\_biblioteca = []  # Añadir libros a la lista  mi\_biblioteca.append(libro1)  mi\_biblioteca.append(libro2)  mi\_biblioteca.append(libro3)  # Mostrar el inventario completo  print("\n\n--- INVENTARIO COMPLETO DE LA BIBLIOTECA ---")  for libro\_actual in mi\_biblioteca:  libro\_actual.mostrar\_info()  print("=" \* 20) # Separador |
|  |
| código: po\_ej3 |

|  |
| --- |
| #Paso 1: Variables Globales  lista\_de\_tareas = []  proximo\_id\_tarea = 1 # Para generar IDs únicos  #Paso 2: Implementar agregar\_tarea  def agregar\_tarea(descripcion, prioridad="media"):  global proximo\_id\_tarea # ¡Necesario para modificar una variable global!  nueva\_tarea = {  "id": proximo\_id\_tarea,  "descripcion": descripcion,  "completada": False,  "prioridad": prioridad  }  lista\_de\_tareas.append(nueva\_tarea)  proximo\_id\_tarea += 1  print(f" Tarea '{descripcion}' añadida con éxito.")  #Paso 3: Implementar mostrar\_tareas  def mostrar\_tareas():  print("\n--- LISTA DE TAREAS ---")  if not lista\_de\_tareas:  print("¡No hay tareas pendientes! ¡A disfrutar!")  return  for tarea in lista\_de\_tareas:  estado = " " if tarea["completada"] else " "  print(f"{estado} ID: {tarea['id']} | {tarea['descripcion']} (Prioridad: {tarea['prioridad']})")  print("------------------------")  #Prueba tus funciones:  agregar\_tarea("Estudiar para el examen de Cálculo")  agregar\_tarea("Hacer las compras", prioridad="alta")  mostrar\_tareas()  #Paso 4: Implementar buscar\_tarea\_por\_id  def buscar\_tarea\_por\_id(id\_buscado):  """Recorre la lista de tareas y devuelve el diccionario de la tarea  que coincide con el id\_buscado. Si no la encuentra, devuelve None."""  for tarea in lista\_de\_tareas:  if tarea["id"] == id\_buscado:  return tarea # ¡Éxito! Devolvemos el diccionario completo  return None # Si el bucle termina, no se encontró  #Prueba tu función de búsqueda:  # Asumiendo que ya agregaste tareas con IDs 1 y 2...  tarea\_encontrada = buscar\_tarea\_por\_id(1)  if tarea\_encontrada:  print(f"\nBúsqueda exitosa: {tarea\_encontrada['descripcion']}")  else:  print("\nBúsqueda fallida: Tarea no encontrada.")  tarea\_fantasma = buscar\_tarea\_por\_id(99)  if not tarea\_fantasma:  print("Búsqueda de tarea inexistente funcionó correctamente.")  #Paso 5: Implementar marcar\_tarea\_completada  def marcar\_tarea\_completada(id\_tarea):  # ¡Reutilizamos nuestra función de búsqueda!  tarea = buscar\_tarea\_por\_id(id\_tarea)  if tarea: # Si la búsqueda devolvió un diccionario (no None)  tarea["completada"] = True  print(f" Tarea '{tarea['descripcion']}' marcada como completada.")  else:  print(f" Error: No se encontró la tarea con ID {id\_tarea}.")  #Paso 6: Implementar eliminar\_tarea  def eliminar\_tarea(id\_tarea):  tarea = buscar\_tarea\_por\_id(id\_tarea)  if tarea:  lista\_de\_tareas.remove(tarea)  print(f" Tarea '{tarea['descripcion']}' eliminada.")  else:  print(f" Error: No se encontró la tarea con ID {id\_tarea}.")  #Prueba todo el flujo:  mostrar\_tareas()  marcar\_tarea\_completada(1)  mostrar\_tareas() # Debería mostrar la tarea 1 como completada  eliminar\_tarea(2)  mostrar\_tareas() # La tarea 2 ya no debería aparecer  marcar\_tarea\_completada(99) # Probar con un ID que no existe  # ... (definiciones de funciones y pruebas aquí arriba) ...  # ¡Puedes comentar o eliminar las pruebas para tener un programa limpio!  while True:  print("\n===== MENÚ TO-DO LIST =====")  print("1. Agregar nueva tarea")  print("2. Mostrar todas las tareas")  print("3. Marcar tarea como completada")  print("4. Eliminar tarea")  print("0. Salir")  opcion = input("Elige una opción: ")  if opcion == '1':  desc = input("Descripción de la nueva tarea: ")  prio = input("Prioridad (alta, media, baja): ")  agregar\_tarea(desc, prio)  elif opcion == '2':  mostrar\_tareas()  elif opcion == '3':  id\_t = int(input("ID de la tarea a completar: "))  marcar\_tarea\_completada(id\_t)  elif opcion == '4':  id\_t = int(input("ID de la tarea a eliminar: "))  eliminar\_tarea(id\_t)  elif opcion == '0':  print("¡Hasta pronto!")  break # Rompe el bucle while  else:  print(" Opción no válida. Inténtalo de nuevo.") |
|  |
| Código: orquestando con un menú principal |

|  |
| --- |
| # =========================  # TO-DO LIST EN PYTHON  # =========================  # Paso 1: Variables Globales  lista\_de\_tareas = []  proximo\_id\_tarea = 1 # Para generar IDs únicos  # Paso 2: Implementar agregar\_tarea  def agregar\_tarea(descripcion, prioridad='media'):  global proximo\_id\_tarea  nueva\_tarea = {  "id": proximo\_id\_tarea,  "descripcion": descripcion,  "completada": False,  "prioridad": prioridad  }  lista\_de\_tareas.append(nueva\_tarea)  proximo\_id\_tarea += 1  print(f"✅ Tarea '{descripcion}' añadida con éxito.")  # Paso 3: Implementar mostrar\_tareas  def mostrar\_tareas():  print("\n--- 📋 LISTA DE TAREAS ---")  if not lista\_de\_tareas:  print("¡No hay tareas pendientes! ¡A disfrutar!")  return  for tarea in lista\_de\_tareas:  estado = "✅" if tarea["completada"] else "⬜"  print(f"{estado} ID: {tarea['id']} | {tarea['descripcion']} (Prioridad: {tarea['prioridad']})")  # Paso 4: Implementar buscar\_tarea\_por\_id  def buscar\_tarea\_por\_id(id\_buscado):  for tarea in lista\_de\_tareas:  if tarea["id"] == id\_buscado:  return tarea  return None  # Paso 5: Implementar marcar\_tarea\_completada  def marcar\_tarea\_completada(id\_tarea):  tarea = buscar\_tarea\_por\_id(id\_tarea)  if tarea:  tarea["completada"] = True  print(f"✅ Tarea '{tarea['descripcion']}' marcada como completada.")  else:  print(f"❌ Error: No se encontró la tarea con ID {id\_tarea}.")  # Paso 6: Implementar eliminar\_tarea  def eliminar\_tarea(id\_tarea):  tarea = buscar\_tarea\_por\_id(id\_tarea)  if tarea:  lista\_de\_tareas.remove(tarea)  print(f"✅ Tarea '{tarea['descripcion']}' eliminada.")  else:  print(f"❌ Error: No se encontró la tarea con ID {id\_tarea}.")  # Paso 7: El Bucle Principal del Programa  while True:  print("\n==== MENÚ TO-DO LIST ====")  print("1. Agregar nueva tarea")  print("2. Mostrar todas las tareas")  print("3. Marcar tarea como completada")  print("4. Eliminar tarea")  print("0. Salir")  opcion = input("Elige una opción: ")  if opcion == '1':  desc = input("Descripción de la nueva tarea: ")  prio = input("Prioridad (alta, media, baja): ")  agregar\_tarea(desc, prio)  elif opcion == '2':  mostrar\_tareas()  elif opcion == '3':  id\_t = int(input("ID de la tarea a completar: "))  marcar\_tarea\_completada(id\_t)  elif opcion == '4':  id\_t = int(input("ID de la tarea a eliminar: "))  eliminar\_tarea(id\_t)  elif opcion == '0':  print("¡Hasta pronto!")  break  else:  print("❌ Opción no válida. Inténtalo de nuevo.") |
|  |
| Código: menú To Do List |

|  |
| --- |
| # Ejercicio 1: Crear y escribir en un archivo de texto llamado "mi\_diario.txt"  # Paso 1: Definimos el nombre del archivo  nombre\_archivo = "mi\_diario.txt"  # Paso 2: Usamos 'with open(...)' en modo escritura ('w')  # Esto Crea el archivo si no existe, y lo sobreescribe si ya existe.  with open(nombre\_archivo, "w") as diario\_file:  # Paso 3: Escribimos varias líneas en el archivo usando .write()  diario\_file.write("Querido diario,\n")  diario\_file.write("Hoy aprendí sobre archivos en Python.\n")  diario\_file.write("El modo 'w' borra todo antes de escribir. ¡Qué miedo!\n")  diario\_file.write("¡Pero también es muy útil para comenzar desde cero!\n")  # Paso 4: Confirmamos que se ha terminado de escribir  print("✅ Diario creado y primeras entradas guardadas.")  # Ejercicio 2: Leer el contenido del diario después de escribirlo  # Opción A: Leer todo de golpe (menos recomendada porque no separa líneas bien al mostrar)  # La dejamos comentada como referencia didáctica  # with open(nombre\_archivo, "r") as diario\_file:  # contenido = diario\_file.read()  # print("\n--- Contenido completo del diario (modo A) ---")  # print(contenido)  # Opción B: Leer línea por línea (más clara y controlada)  print("\n--- Contenido del diario (línea por línea) ---")  try:  with open(nombre\_archivo, "r") as diario\_file:  for linea in diario\_file:  print(linea.strip()) # .strip() elimina el salto de línea al final  except FileNotFoundError:  print(f"❌ Error: El archivo '{nombre\_archivo}' no existe.")  # Ejercicio 2: Leer el contenido del archivo "mi\_diario.txt"  # Probamos dos formas: opción A (leer todo) y opción B (leer línea por línea)  # ------------------------------  # 🔹 Opción A: Leer todo de golpe  # ------------------------------  # Esta opción carga todo el contenido del archivo como un solo string  # Útil si queremos procesar o mostrar todo junto  print("\n--- Contenido completo del diario (modo A - read()) ---")  try:  with open(nombre\_archivo, "r") as diario\_file:  contenido = diario\_file.read() # Lee todo el contenido de una sola vez  print(contenido)  except FileNotFoundError:  print(f"❌ Error: El archivo '{nombre\_archivo}' no existe.")  # -----------------------------------  # 🔸 Opción B: Leer línea por línea  # -----------------------------------  # Esta opción permite manejar cada línea por separado  # Ideal para procesar o mostrar contenido ordenado, sin saltos extra  print("\n--- Contenido del diario (modo B - línea por línea) ---")  try:  with open(nombre\_archivo, "r") as diario\_file:  for linea in diario\_file:  print(linea.strip()) # Eliminamos los '\n' del final para una impresión limpia  except FileNotFoundError:  print(f"❌ Error: El archivo '{nombre\_archivo}' no existe.")  # Ejercicio 3: Añadir nuevas entradas al archivo sin borrar lo anterior (modo 'a')  print("\n📝 Añadiendo nuevas entradas al diario...")  # Abrimos el archivo en modo añadir ('a')  with open(nombre\_archivo, "a") as diario\_file:  # Escribimos nuevas líneas. También podemos añadir una línea separadora.  diario\_file.write("\n--- Entrada del 20 de Junio de 2025 ---\n")  diario\_file.write("El modo 'a' es genial para no perder datos.\n")  diario\_file.write("Ahora mi diario puede crecer cada día.\n")  # Confirmamos que se añadieron las nuevas entradas  print("✅ Nuevas entradas guardadas.")  # Verificamos que las nuevas entradas se añadieron correctamente  print("\n📖 Verificando el contenido final del diario...")  try:  with open(nombre\_archivo, "r") as diario\_file:  for linea in diario\_file:  print(linea.strip())  except FileNotFoundError:  print(f"❌ Error: El archivo '{nombre\_archivo}' no existe.") |
|  |
| Código: Diario, nuevas entradas |

|  |
| --- |
|  |
|  |
| Código: |

|  |
| --- |
|  |
|  |
| Código: |