****

**TEMA: CLASES Y OBJETOS**

**NOMBRE DEL ALUMNO:** Woolfolk Cerecer Brian

**SEMESTRE:** 8vo

**NOMBRE DE LA MATERIA:** Seminario de programación

**CLAVE DE LA MATERIA**: COM35C1

**INTRODUCCIÓN**

A continuación, se presenta una solución de software elaborado con el lenguaje de programación Python, para la administración sencilla de una biblioteca de libros.

El sistema tiene como objetivo final permitir la búsqueda y filtrado de información mediante título del libro, autor del libro o sección del libro. Este problema supone crear al menos tres clases distintas que modelen la información y estructura de un libro, de un autor y de la biblioteca como tal. Se consideró innecesario generar una clase para “secciones”, puesto el único valor a almacenar será su nombre (ej. “Misterio”, “Aventuras”, “Ciencia ficción”, entre otros).

Esta información se almacenará como propiedades de la biblioteca, estructuradas en una lista de datos (lista de libros, lista de autores y lista de cadenas de texto para las secciones, según lo mencionado previamente). Así mismo, se identifica una clara relación entre la clase Libro y Autor, por lo cual se destinará una propiedad para los libros llamada “autor\_id” que guarde relación con la lista de autores existentes en la biblioteca global (mismo caso para las secciones, a través de la propiedad “seccion\_id”).

Las propiedades de la clase biblioteca serán únicamente de lectura, pues no es necesario volver a generar las listas, sino mejor modificarlas a través de métodos especializados para ello. Las posibles acciones del sistema serán para crear, reemplazar, leer y eliminar los datos (es decir, las listas de información).

Al tratarse de muchos procesos y muy similares entre ellos, se crearán funciones que faciliten la lectura y escritura de datos para el usuario, teniendo en consideración el manejo de errores y correcto tipado de las variables. Cabe mencionar que las clases de Libro y Autor necesitarán forzosamente de la intervención del usuario para completar sus campos, esto con el fin de garantizar la captura correcta y evitar la inyección de código y tipos de datos erróneos.

No es considerado un problema el hecho de que el usuario intervenga completamente en la captura de datos, sino al contrario, pues este sistema partirá de una base de datos inexistente y toda información es requerida de primera mano por parte del usuario (utilizando los métodos y funciones ya conocidas como input(), manejo de errores con cláusulas try-except y ciclos while para insistir en la captura de un valor válido para el sistema).

A fin de mantener un correcto manejo de errores y datos, se consultó la documentación oficial de Python al respecto de la declaración explícita de tipos de datos, tanto para crear estructuras nuevas, clases y para funciones (tanto para parámetros como para valores de retorno).

**DESARROLLO**

|  |
| --- |
| **Código** |
| # -------------------------------- CLASE BIBLIOTECA  # PARA GUARDAR LOS LIBROS, SECCIONES Y AUTORES  class Biblioteca:  # -------------------------------- METODO INICIALIZADOR  def \_\_init\_\_(self):  # AL COMIENZO LA BIBLIOTECA ESTARA VACIA COMPLETAMENTE  # INDICAMOS LOS TIPOS DE LAS LISTAS, AUNQUE ESTEN VACIAS  self.\_libros: list[Libro] = []  self.\_secciones: list[str] = []  self.\_autores: list[Autor] = []  # -------------------------------- METODO PARA MOSTRAR COMO CADENA  def \_\_str\_\_(self):  # MOSTRAMOS LA INFORMACION  self.ver\_secciones()  self.ver\_autores()  self.ver\_libros()  # REGRESAMOS UN STRING POR DEFAULT  return "Biblioteca"  # -------------------------------- PROPIEDADES DE LA CLASE (SOLO LECTURA)  # ================ LIBROS  @property  def libros(self):  return self.\_libros  # ================ SECCIONES  @property  def secciones(self):  return self.\_secciones  # ================ AUTORES  @property  def autores(self):  return self.\_autores  # -------------------------------- METODO PARA MOSTRAR LAS SECCIONES  def ver\_secciones(self):  print("Secciones existentes:")  mostrar\_lista(self.secciones)  # -------------------------------- METODO PARA MOSTRAR LOS AUTORES  def ver\_autores(self):  print("Autores existentes:")  mostrar\_lista(self.autores)  # -------------------------------- METODO PARA MOSTRAR LOS LIBROS  def ver\_libros(self):  print("Libros existentes:")  mostrar\_lista(self.libros)  # -------------------------------- METODO CREAR LIBROS  def crear\_libro(self):  # VERIFICAR SI EXISTEN SECCIONES  if len(self.secciones) == 0:  print("--- Es necesario agregar secciones primero!\n")  return # SI NO HAY CANCELAMOS LA CREACION  # VERIFICAR SI EXISTEN AUTORES  if len(self.autores) == 0:  print("--- Es necesario agregar autores primero!\n")  return # SI NO HAY CANCELAMOS LA CREACION  # CREAR EL LIBRO (EL CONSTRUCTOR CAPTURA LA INFORMACION)  nuevo\_libro = Libro()  self.libros.append(nuevo\_libro) # AGREGAMOS EL LIBRO  print("\n Libro creado con exito!\n")  # -------------------------------- METODO CREAR SECCIONES  def crear\_secciones(self):  # UTILIZAMOS LA FUNCION leer\_string PARA GARANTIZAR UNA CADENA VALIDA  nueva\_seccion = leer\_string("Nueva seccion: ")  self.secciones.append(nueva\_seccion) # AGREGAMOS LA SECCION  print("\n Seccion creada con exito!\n")  # -------------------------------- METODO CREAR AUTORES  def crear\_autor(self):  # CREAR EL AUTOR (EL CONSTRUCTOR CAPTURA LA INFORMACION)  nuevo\_autor = Autor()  self.autores.append(nuevo\_autor) # AGREGAMOS AL AUTOR  print("\n Autor creado con exito!\n")  # -------------------------------- METODO ELIMINAR LIBROS  def eliminar\_libro(self):  # SOLICITAMOS AL USUARIO EL INDICE A ELIMINAR (seleccionar\_indice)  eliminar = seleccionar\_indice("Elige el libro a eliminar: ", self.libros)  # REVISAMOS SI LA LISTA ESTA VACIA  if eliminar == -1:  return # CANCELAMOS LA ELIMINACION  try:  # ELIMINAMOS EL INDICE SELECCIONADO  del self.libros[eliminar]  print("\n Libro eliminado con exito!\n")  except Exception:  # 'ATRAPAMOS' CUALQUIER ERROR POR LA ELIMINACION  print("--- Error inesperado!\n")  # -------------------------------- METODO ELIMINAR SECCIONES  def eliminar\_seccion(self):  # SOLICITAMOS AL USUARIO EL INDICE A ELIMINAR (seleccionar\_indice)  eliminar = seleccionar\_indice("Elige la seccion a eliminar: ", self.secciones)  # REVISAMOS SI LA LISTA ESTA VACIA  if eliminar == -1:  return # CANCELAMOS LA ELIMINACION  try:  i = 0 # CONTADOR  # CICLAMOS MIENTRAS EL CONTADOR NO LLEGUE AL FINAL  while i < len(self.libros):  # SI EL LIBRO ACTUAL ES DE LA SECCION QUE ESTAMOS ELIMINANDO...  if self.libros[i].seccion\_id == eliminar:  del self.libros[i] # ...TAMBIEN SE BORRA  i -= 1 # EVITAMOS AVANZAR EL CONTADOR  elif self.libros[i].seccion\_id > eliminar:  # SI SON DE UNA SECCION MAS ADELANTE...  self.libros[i].seccion\_id -= 1 # ...LO RECORREMOS  i += 1 # AUMENTAMOS EL CONTADOR  # ELIMINAMOS EL INDICE SELECCIONADO  del self.secciones[eliminar]  print("\n Seccion eliminada con exito!\n")  except Exception:  # 'ATRAPAMOS' CUALQUIER ERROR POR LA ELIMINACION  print("--- Error inesperado!\n")  # -------------------------------- METODO ELIMINAR AUTORES  def eliminar\_autores(self):  # SOLICITAMOS AL USUARIO EL INDICE A ELIMINAR (seleccionar\_indice)  eliminar = seleccionar\_indice("Elige el autor a eliminar: ", self.autores)  # REVISAMOS SI LA LISTA ESTA VACIA  if eliminar == -1:  return # CANCELAMOS LA ELIMINACION  try:  i = 0 # CONTADOR  # CICLAMOS MIENTRAS EL CONTADOR NO LLEGUE AL FINAL  while i < len(self.libros):  # SI EL LIBRO ACTUAL ES DE UN AUTOR QUE ESTAMOS ELIMINANDO...  if self.libros[i].autor\_id == eliminar:  del self.libros[i] # ...TAMBIEN SE BORRA  i -= 1 # EVITAMOS AVANZAR EL CONTADOR  elif self.libros[i].autor\_id > eliminar:  # SI SON DE UN AUTOR MAS ADELANTE...  self.libros[i].autor\_id -= 1 # ...LO RECORREMOS  i += 1 # AUMENTAMOS EL CONTADOR  # ELIMINAMOS EL INDICE SELECCIONADO  del self.autores[eliminar]  print("\n Autor eliminado con exito!\n")  except Exception:  # 'ATRAPAMOS' CUALQUIER ERROR POR LA ELIMINACION  print("--- Error inesperado!\n")  # -------------------------------- METODO REEMPLAZAR LIBROS  def reemplazar\_libro(self):  # SOLICITAMOS AL USUARIO EL INDICE A REEMPLAZAR (seleccionar\_indice)  reemplazar = seleccionar\_indice("Elige el libro a reemplazar: ", self.libros)  # REVISAMOS SI LA LISTA ESTA VACIA  if reemplazar == -1:  return # CANCELAMOS EL REEMPLAMIENTO  print("") # SALTO DE LINEA  # CREAMOS UN NUEVO LIBRO PARA REEMPLAZAR  self.libros[reemplazar] = Libro()  print("\n Libro reemplazado con exito!\n")  # -------------------------------- METODO REEMPLAZAR SECCIONES  def reemplazar\_seccion(self):  # SOLICITAMOS AL USUARIO EL INDICE A REEMPLAZAR (seleccionar\_indice)  reemplazar = seleccionar\_indice("Elige la seccion a reemplazar: ", self.secciones)  # REVISAMOS SI LA LISTA ESTA VACIA  if reemplazar == -1:  return # CANCELAMOS EL REEMPLAZAMIENTO  print("") # SALTO DE LINEA  # CREAMOS UNA NUEVA SECCION PARA REEMPLAZAR  self.secciones[reemplazar] = leer\_string("Ingresa la nueva seccion: ")  print("\n Seccion reemplazada con exito!\n")  # -------------------------------- METODO REEMPLAZAR AUTORES  def reemplazar\_autores(self):  # SOLICITAMOS AL USUARIO EL INDICE A REEMPLAZAR (seleccionar\_indice)  reemplazar = seleccionar\_indice("Elige el autor a reemplazar: ", self.autores)  # REVISAMOS SI LA LISTA ESTA VACIA  if reemplazar == -1:  return # CANCELAMOS EL REEMPLAZAMIENTO  print("") # SALTO DE LINEA  # CREAMOS UN NUEVO AUTOR PARA REEMPLAZAR  self.autores[reemplazar] = Autor()  print("\n Autor reemplazado con exito!\n")  # -------------------------------- METODO PARA FILTRAR LIBROS  def filtrar\_libros(self):  print("--- BUSQUEDA ---\n")  # REVISAMOS SI HAY LIBROS PARA FILTRAR  if len(self.libros) == 0:  print(" No hay libros para filtrar!\n")  return # CANCELAMOS LA BUSQUEDA  # CREAMOS EL FILTRO DEL TITULO, LO MARCAMOS VACIO,  # PERO INDICAMOS QUE TAMBIEN PUEDE CONTENER UNA CADENA  buscar\_titulo: str | None = None  # PREGUNTAMOS AL USUARIO SI QUIERE BUSCAR POR NOMBRE DEL LIBRO  if leer\_respuesta("Desea buscar por titulo? SI/NO: "):  # EN CASO AFIRMATIVO, SOLICITAMOS ENTONCES EL TITULO DE BUSQUEDA  buscar\_titulo = leer\_string("> Fragmento del titulo del libro: ")  print("") # SALTO DE LINEA  # CREAMOS EL FILTRO DE LA SECCION, LO MARCAMOS VACI0  buscar\_seccion: int | None = None  # PREGUNTAMOS AL USUARIO SI QUIERE BUSCAR POR SECCION  if leer\_respuesta("Desea buscar por seccion? SI/NO: "):  # EN CASO AFIRMATIVO, SOLICITAMOS ENTONCES LA SECCION (seleccionar\_indice)  buscar\_seccion = seleccionar\_indice("> Escoge una seccion: ", self.secciones)  print("") # SALTO DE LINEA  # CREAMOS EL FILTRO DEL AUTOR, LO MARCAMOS VACIO  buscar\_autor: int | None = None  # PREGUNTAMOS AL USUARIO SI QUIERE BUSCAR POR NOMBRE DEL AUTOR  if leer\_respuesta("Desea buscar por autor? SI/NO: "):  # EN CASO AFIRMATIVO, SOLICITAMOS ENTONCES AL AUTOR (seleccionar\_indice)  buscar\_autor = seleccionar\_indice("> Escoge un autor: ", self.autores)  print("") # SALTO DE LINEA  # AHORA QUE TENEMOS LOS FILTROS, MOSTRAREMOS SOLO LO QUE COINCIDA  print("\n--- Libros encontrados ---\n")  i = 0 # INICIAMOS UN CONTADOR PARA EL SIGUIENTE CICLO...  for libro in self.libros:  # BANDERA QUE NOS AYUDA A SUMAR BOOLEANOS  mostrar = True  # SI EXISTE UN FILTRO DE TITULO, LO APLICAMOS  if buscar\_titulo is not None:  mostrar = buscar\_titulo.lower() in libro.titulo.lower()  # SI EXISTE UN FILTRO DE SECCION, LO APLICAMOS  if buscar\_seccion is not None:  mostrar = mostrar and (buscar\_seccion == libro.seccion\_id)  # SI EXISTE UN FILTRO DE AUTOR, LO APLICAMOS  if buscar\_autor is not None:  mostrar = mostrar and (buscar\_autor == libro.autor\_id)  # AL HABER PASADO POR LOS FILTROS EVALUAMOS SI PODEMOS MOSTRAR  if mostrar:  print(f" {i + 1}. {libro}")  i += 1 # AVANZAMOS EL CONTADOR  # EN CASO DE QUE NINGUN LIBRO SE MOSTRO  if i == 0:  print(" No hay libros que cumplan los requisitos!")  print("") # SALTO DE LINEA  # -------------------------------- CLASE LIBRO  # QUE CONTENGA TITULO, PAGINAS Y AÑO,  # ASI COMO EL ID DEL AUTOR Y LA SECCION A LA QUE PERTENEZCA  class Libro:  # -------------------------------- METODO INICIALIZADOR  def \_\_init\_\_(self):  # PEDIR AL USUARIO LOS CAMPOS INICIALES  print("=== Crear libro:\n")  self.\_titulo = leer\_string("> Titulo: ")  self.\_paginas = 1 # PRIMERO CREAMOS LA PROPIEDAD, LUEGO LA LLAMAMOS  self.paginas = leer\_entero("> Paginas: ", True)  self.\_anio = 1 # PRIMERO CREAMOS LA PROPIEDAD, LUEGO LA LLAMAMOS  self.anio = leer\_entero("> Anio de publicacion: ", True)  # INDICAMOS AL USUARIO QUE SELECCIONE DE ENTRE TODOS LOS AUTORES DISPONIBLES  self.\_autor\_id = seleccionar\_indice("> Autor: ", biblioteca.autores)  # INDICAMOS AL USUARIO QUE SELECCIONE DE ENTRE TODAS LAS SECCIONES DISPONIBLES  self.\_seccion\_id = seleccionar\_indice("> Seccion: ", biblioteca.secciones)  # -------------------------------- METODO PARA MOSTRAR COMO CADENA  def \_\_str\_\_(self):  try:  # BUSCAMOS AL AUTOR DEL LIBRO USANDO autor\_id  autor = biblioteca.autores[self.autor\_id]  # BUSCAMOS LA SECCION DEL LIBRO USANDO seccion\_id  seccion = biblioteca.secciones[self.seccion\_id]  # MOSTRAMOS LA INFORMACION  return f"{self.titulo} - {autor.nombre} ({self.anio}) [{seccion}]"  except Exception:  # EN CASO DE QUE EL AUTOR O SECCION NO EXISTAN, MOSTRAMOS  return f"{self.titulo} ({self.anio})"  # -------------------------------- PROPIEDADES DE LA CLASE (LECTURA Y ESCRITURA)  # ================ TITULO  @property  def titulo(self):  return self.\_titulo  @titulo.setter  def titulo(self, valor: str):  nuevo\_valor = string\_valido(valor)  if nuevo\_valor is not None:  self.\_titulo = nuevo\_valor  # ================ PAGINAS  @property  def paginas(self):  return self.\_paginas  @paginas.setter  def paginas(self, valor: int):  nuevo\_valor = entero\_valido(valor, True)  if nuevo\_valor is not None:  if nuevo\_valor == 0:  # CONSIDERAMOS AL 0 COMO 1  self.\_paginas = 1  else:  self.\_paginas = nuevo\_valor  # ================ ANIO  @property  def anio(self):  return self.\_anio  @anio.setter  def anio(self, valor: int):  nuevo\_valor = entero\_valido(valor, True)  if nuevo\_valor is not None:  if nuevo\_valor == 0:  # CONSIDERAMOS AL 0 COMO 1  self.\_anio = 1  else:  self.\_anio = nuevo\_valor  # ================ AUTOR\_ID  @property  def autor\_id(self):  return self.\_autor\_id  @autor\_id.setter  def autor\_id(self, valor: int):  nuevo\_valor = indice\_valido(valor, len(biblioteca.autores))  if nuevo\_valor is not None:  self.\_autor\_id = nuevo\_valor  # ================ SECCION\_ID  @property  def seccion\_id(self):  return self.\_seccion\_id  @seccion\_id.setter  def seccion\_id(self, valor: int):  nuevo\_valor = indice\_valido(valor, len(biblioteca.secciones))  if nuevo\_valor is not None:  self.\_seccion\_id = nuevo\_valor  # -------------------------------- CLASE AUTOR  # QUE CONTENGA SU NOMBRE, FECHA DE NACIMIENTO Y NACIONALIDAD  class Autor:  # -------------------------------- METODO INICIALIZADOR  def \_\_init\_\_(self):  # PEDIR AL USUARIO LOS CAMPOS INICIALES  print("=== Crear autor:\n")  self.\_nombre = leer\_string("> Nombre: ")  self.\_nacimiento = 1 # PRIMERO CREAMOS LA PROPIEDAD, LUEGO LA LLAMAMOS  self.nacimiento = leer\_entero("> Anio de Nacimiento: ", True)  self.\_nacionalidad = leer\_string("> Nacionalidad: ")  # -------------------------------- METODO PARA MOSTRAR COMO CADENA  def \_\_str\_\_(self):  # MOSTRAMOS CON EL SIGUIENTE FORMATO  return f"{self.nombre} ({self.nacimiento}, {self.nacionalidad})"  # -------------------------------- PROPIEDADES DE LA CLASE  # ================ NOMBRE  @property  def nombre(self):  return self.\_nombre  @nombre.setter  def nombre(self, valor: str):  nuevo\_valor = string\_valido(valor)  if nuevo\_valor is not None:  self.\_nombre = nuevo\_valor  # ================ NACIMIENTO  @property  def nacimiento(self):  return self.\_nacimiento  @nacimiento.setter  def nacimiento(self, valor: int):  nuevo\_valor = entero\_valido(valor, True)  if nuevo\_valor is not None:  if nuevo\_valor == 0:  # CONSIDERAMOS AL 0 COMO 1  self.\_nacimiento = 1  else:  self.\_nacimiento = nuevo\_valor  # ================ NACIONALIDAD  @property  def nacionalidad(self):  return self.\_nacionalidad  @nacionalidad.setter  def nacionalidad(self, valor: int):  nuevo\_valor = string\_valido(valor)  if nuevo\_valor is not None:  self.\_nacionalidad = nuevo\_valor  # -------------------------------- FUNCIONES DE COMPROBACION DE TIPOS  # REVISA SI EL STRING ES VALIDO Y LO REGRESA, SINO REGRESA None  def string\_valido(valor) -> str | None:  if not isinstance(valor, str) or valor == "":  print("--- Ingrese una cadena de texto valida!\n")  return None  return valor # ESTE VALOR ES VALIDO  # REVISA SI LA RESPUESTA ES 'SI/NO' Y REGRESA True/False, SINO REGRESA None  def respuesta\_valida(valor) -> bool | None:  if not isinstance(valor, str):  print("--- Ingrese 'SI' o 'NO'\n")  return None  if valor.lower() == "si":  return True # ESTE VALOR ES VALIDO  if valor.lower() == "no":  return False # ESTE VALOR ES VALIDO  print("--- Ingrese 'SI' o 'NO'\n")  return None # CUALQUIER OTRO RESULTADO  # REVISA SI EL ENTERO ES VALIDO Y LO REGRESA, SINO REGRESA None,  # TAMBIEN NOS PERMITE INDICAR SI QUEREMOS QUE SEA POSITIVO  def entero\_valido(valor, es\_positivo: bool) -> int | None:  if not isinstance(valor, int) or (es\_positivo and valor < 0):  if es\_positivo:  print("--- Ingrese un numero entero positivo!\n")  else:  print("--- Ingrese un numero entero!\n")  return None  return valor # ESTE VALOR ES VALIDO  # REVISA SI UN NUMERO REPRESENTA UN INDEX DE UNA LISTA  # EL PARAMETRO tamanio\_maximo REPRESENTA EL len(lista)  def indice\_valido(valor, tamanio\_maximo: int) -> int | None:  # EN CASO DE QUE LA LISTA ESTE VACIA, REGRESAMOS -1  if tamanio\_maximo == 0:  print("--- Esta vacio!\n")  return -1 # GRACIAS AL RESTO DE VERIFICACIONES, ESTO PREVIENE ERRORES  # VERIFICAMOS SI EL NUMERO ES VALIDO PRIMERO  valor = entero\_valido(valor, True)  if valor is None:  return None # SIGNIFICA QUE EL NUMERO NO ERA VALIDO  # DESPUES, VERIFICAMOS SI EL NUMERO ENTRA DENTRO DEL RANGO VALIDO  if not (tamanio\_maximo > valor >= 0):  print("--- Ingrese un numero valido de la lista!\n")  return None # NO EXISTE EN LA LISTA  return valor # ESTE VALOR ES VALIDO  # -------------------------------- FUNCIONES DE LECTURA RAPIDA  # FUNCION PARA OBTENER UNA CADENA DE TEXTO VALIDA 100%  def leer\_string(mensaje: str) -> str:  # CREAMOS UN CICLO 'INFINITO' PARA INSISTIR CON LA CAPTURA  while True:  try:  # SOLICITAMOS EL VALOR Y LO COMPROBAMOS  valor = string\_valido(input(mensaje))  # SI LA COMPROBACION FUNCIONA, REGRESAMOS EL VALOR  if valor is not None:  return valor # CIERRA EL CICLO CON UN VALOR CORRECTO  # NO ES NECESARIO MOSTRAR ERRORES GRACIAS A string\_valido  # continue  except ValueError:  # 'ATRAPAMOS' CUALQUIER ERROR INESPERADO Y REINICIAMOS EL CICLO  print("--- Ingrese una cadena de texto valida!\n")  # FUNCION PARA OBTENER UNA RESPUESTA DE 'SI/NO' (bool) 100%  def leer\_respuesta(mensaje: str) -> bool:  # CREAMOS UN CICLO 'INFINITO' PARA INSISTIR CON LA CAPTURA  while True:  try:  # SOLICITAMOS EL VALOR Y LO COMPROBAMOS  valor = respuesta\_valida(input(mensaje))  # SI LA COMPROBACION FUNCIONA, REGRESAMOS EL VALOR  if valor is not None:  return valor # CIERRA EL CICLO CON UN VALOR CORRECTO  # NO ES NECESARIO MOSTRAR ERRORES GRACIAS A respuesta\_valida  # continue  except ValueError:  # 'ATRAPAMOS' CUALQUIER ERROR INESPERADO Y REINICIAMOS EL CICLO  print("--- Ingrese 'SI' o 'NO'\n")  # FUNCION PARA OBTENER UN NUMERO ENTERO VALIDO 100%,  # PUDIENDO DECIR SI QUEREMOS QUE SEA POSITIVO O NO  def leer\_entero(mensaje: str, es\_positivo: bool) -> int:  # CREAMOS UN CICLO 'INFINITO' PARA INSISTIR CON LA CAPTURA  while True:  try:  # SOLICITAMOS EL VALOR, LO CONVERTIMOS A int Y LO COMPROBAMOS  valor = entero\_valido(int(input(mensaje)), es\_positivo)  # SI LA COMPROBACION FUNCIONA, REGRESAMOS EL VALOR  if valor is not None:  return valor # CIERRA EL CICLO CON UN VALOR CORRECTO  # NO ES NECESARIO MOSTRAR ERRORES GRACIAS A entero\_valido  # continue  except ValueError:  # 'ATRAPAMOS' CUALQUIER ERROR INESPERADO Y REINICIAMOS EL CICLO  print("--- Ingrese un numero valido!\n")  # FUNCION PARA ESCOGER ENTRE UNA LISTA DE OPCIONES (Y TAMBIEN MUESTRA LAS OPCIONES)  def seleccionar\_indice(mensaje: str, lista: list) -> int:  # EN CASO DE QUE LA LISTA ESTE VACIA, REGRESAMOS -1  if len(lista) == 0:  print("--- Esta vacio!\n")  return -1 # GRACIAS AL RESTO DE VERIFICACIONES, ESTO PREVIENE ERRORES  # CREAMOS UN CICLO 'INFINITO' PARA INSISTIR CON LA CAPTURA  while True:  try:  # PRIMERO MOSTRAMOS LAS OPCIONES  print("Escriba el numero de la opcion que desee:")  mostrar\_lista(lista) # USAMOS mostrar\_lista  # DESPUES SOLICITAMOS EL VALOR Y LO VERIFICAMOS  # RESTAMOS 1 AL VALOR PORQUE LOS INDICES MOSTRADOS COMIENZAN CON 1  valor = indice\_valido(int(input(mensaje)) - 1, len(lista))  # SI LA COMPROBACION FUNCIONA, REGRESAMOS EL VALOR  if valor is not None:  return valor # CIERRA EL CICLO CON UN VALOR CORRECTO  # NO ES NECESARIO MOSTRAR ERRORES GRACIAS A indice\_valido  # continue  except ValueError:  # 'ATRAPAMOS' CUALQUIER ERROR INESPERADO Y REINICIAMOS EL CICLO  print("--- Ingrese un numero valido de la lista!\n")  # FUNCION PARA MOSTRAR UNA LISTA JUNTO A SUS INDICES  def mostrar\_lista(lista: list):  indice = 0 # CREAMOS UN CONTADOR  for item in lista:  # MOSTRAMOS EL INDICE Y EL VALOR  # EL INDICE SUMA 1 PARA CONTAR CON NUMEROS NATURALES  print(f" {indice + 1}. {item}")  indice += 1 # AVANZAMOS  # SI EL INDICE NUNCA CAMBIA, ES QUE NO HAY ITEMS EN LA LISTA  if indice == 0:  print(" Esta vacio!")  print("") # SALTO DE LINEA  # -------------------------------- COMENZAR EL PROGRAMA  print("=== BIBLIOTECA EN PYTHON ===\n")  # INICIALIZAMOS LA BIBLIOTECA COMO GLOBAL  biblioteca = Biblioteca()  # MENU DE OPCIONES PARA FACIL IMPRESION  temas = [  "Consultar libros", # TEMA 0  "Ir a Secciones", # TEMA 1  "Ir a Autores", # TEMA 2  "Ir a Libros", # TEMA 3  "Ver todos los datos", # TEMA 4  "Salir" # TEMA 5  ]  # MENU DE ACCIONES PARA FACIL IMPRESION  acciones = [  "Ver todo", # ACCION 0  "Crear nuevo", # ACCION 1  "Reemplazar", # ACCION 2  "Eliminar", # ACCION 3  "Elegir otro tema" # ACCION 4  ]  # -------------------------------- CICLO PRINCIPAL PARA EVALUAR OPCIONES  while True:  print("MENU PRINCIPAL")  # MOSTRAMOS EL MENU PRINCIPAL UTILIZANDO seleccionar\_indice CON temas  tema = seleccionar\_indice("Seleccione un tema: ", temas)  print("========\n")  if tema == 0:  # CONSULTAR LIBROS  biblioteca.filtrar\_libros() # INICIAMOS LA BUSQUEDA  # TIEMPO DE ESPERA PARA VER RESULTADOS (MENU PRINCIPAL)  input("Pulse 'ENTER' para regresar al menu principal... ")  elif tema == 4:  # CONSULTAR TODA LA INFORMACION  biblioteca.\_\_str\_\_() # ESTE METODO IMPRIME EN CONSOLA  # TIEMPO DE ESPERA PARA VER RESULTADOS (MENU PRINCIPAL)  input("Pulse 'ENTER' para regresar al menu principal... ")  elif tema == 5:  # OPCION 5 PARA SALIR  print(" Saliendo...")  break # ROMPEMOS EL CICLO PRINCIPAL Y TERMINA EL PROGRAMA  # ---------------- CICLO SECUNDARIO PARA EL MENU DE ACCIONES  # PASAMOS AL SEGUNDO CICLO DEL SUB-MENU  while True:  if tema == 1:  # ADMINISTRAR SECCIONES  print("> MENU DE SECCIONES: ")  # MOSTRAMOS EL MENU SECUNDARIO UTILIZANDO seleccionar\_indice CON acciones  accion = seleccionar\_indice("Seleccione una accion: ", acciones)  print("") # SALTO DE LINEA  # REACCIONAMOS SEGUN LA ACCION  if accion == 0:  biblioteca.ver\_secciones()  elif accion == 1:  biblioteca.crear\_secciones()  elif accion == 2:  biblioteca.reemplazar\_seccion()  elif accion == 3:  biblioteca.eliminar\_seccion()  else:  break # SALIMOS DEL CICLO SECUNDARIO  elif tema == 2:  # ADMINISTRAR AUTORES  print("> MENU DE AUTORES: ")  # MOSTRAMOS EL MENU SECUNDARIO UTILIZANDO seleccionar\_indice CON acciones  accion = seleccionar\_indice("Seleccione una accion: ", acciones)  print("") # SALTO DE LINEA  # REACCIONAMOS SEGUN LA ACCION  if accion == 0:  biblioteca.ver\_autores()  elif accion == 1:  biblioteca.crear\_autor()  elif accion == 2:  biblioteca.reemplazar\_autores()  elif accion == 3:  biblioteca.eliminar\_autores()  else:  break # SALIMOS DEL CICLO SECUNDARIO  elif tema == 3:  # ADMINISTRAR LIBROS  print("> MENU DE LIBROS: ")  # MOSTRAMOS EL MENU SECUNDARIO UTILIZANDO seleccionar\_indice CON acciones  accion = seleccionar\_indice("Seleccione una accion: ", acciones)  print("") # SALTO DE LINEA  # REACCIONAMOS SEGUN LA ACCION  if accion == 0:  biblioteca.ver\_libros()  elif accion == 1:  biblioteca.crear\_libro()  elif accion == 2:  biblioteca.reemplazar\_libro()  elif accion == 3:  biblioteca.eliminar\_libro()  else:  break # SALIMOS DEL CICLO SECUNDARIO  else:  break # SALIDA DEFAULT, PARA EVITAR CAER EN CICLO INFINITO  # TIEMPO DE ESPERA PARA VER RESULTADOS (SUB MENU)  input("Pulse 'ENTER' para continuar... ")  print("========\n") # SEPARADOR  print("========\n") # SEPARADOR |

|  |
| --- |
| **Visualización de los resultados del código** |
| Imagen 1. Menú principal del programa  Imagen 2. Acceder al submenú de Secciones, donde además se creó una nueva sección llamada “Misterio”  Este último paso se repitió 2 veces más para ingresar las secciones “Terror” y “Fantasía”    Imagen 3. Secciones existentes utilizando el submenú secciones    Imagen 4. Acceder al submenú de Autores, desde el cual se ingresará un nuevo autor.    Imagen 5. Crear un nuevo autor en el sistema  Así mismo, se ingresaron 2 autores más: “Miguel de Cervantes” y “Franz Kafka”    Imagen 6. Mostrar autores existentes    Imagen 7. Nos dirigimos al submenú de Libros para ingresar uno nuevo.    Imagen 8. Captura de datos para un nuevo libro (NOTA: los campos fueron llenados con datos sencillos a fin de agilizar el proceso).  Este último paso se realizó 5 veces más, a fin de crear dos libros para cada autor y para cada sección, alternando entre éstos.    Imagen 9. Mostrar los libros ingresados en el sistema    Imagen 10. Mostrar información capturada hasta el momento, utilizando la opción no.5 del menú principal.    Imagen 11. Iniciar el proceso búsqueda de libros por filtrado.    Imagen 12. Ejemplo de búsqueda de libros por sección “Misterio”.  Debido a la complejidad y al alcance del programa, se omitieron las funcionalidades de Reemplazo, Eliminación y Filtros en este documento. |

**CONCLUSIÓN**

Durante la realización de esta actividad, se propuso crear una estructura modular que sirva como base firme para futuras actividades, fruto de esto podemos encontrar a las funciones de lectura y comprobación de errores como leer\_string, leer\_respuesta y leer\_entero, pero sobre todo con funciones como seleccionar\_indice y mostrar\_lista, ya que éstas dos trabajan en conjunto para brindar funcionalidad total al sistema, desde la selección del menú de opciones, hasta la identificación de índices de lista para relacionar las clases entre sí (como es el caso de Libro, que guarda relación con Autor y la lista de secciones con la propiedad autor\_id y seccion\_id.

A pesar de disponer de una elevada cantidad de líneas de código, esto también demuestra el grado de robustez y complejidad de ciertos sistemas que muchas veces damos por alto como “algo de rutina”. Cabe aclarar que, de no ser por las múltiples funciones generadas, el programa podría alcanzar el doble de líneas de código, y sería entonces más propenso a “errores de dedo” (sin mencionar la dificultad de modificar ciertas secciones de código).

Finalmente, un aspecto indudable a mejorar del sistema es permitir la modificación directa de las propiedades y atributos de las clases, es decir, de los Libros y Autores. El sistema se limita a permitir el *reemplazo* de valores, pero no la *modificación* como tal, esta decisión fue en pro de mantener reducido la (aún así) extensa complejidad del programa sin dejar la posibilidad de cambios directos. También hace falta una verificación adicional para evitar ingresar datos duplicados al sistema, lo cual en ciertos casos puede considerarse un error. Debido a la naturaleza de los datos (objetos, listas y diccionarios) resulta *incómodo* verificar exhaustivamente si cada campo de cada propiedad se encuentra duplicado, por lo que se dejó de lado en esta actividad, pero sin duda es una medida de seguridad muy importante a tomar en cuenta para sistemas que utilicen y accedan a una base de datos formal.

Así mismo, el utilizar la consola para ejecutar el programa de Python limita la interacción del usuario con el sistema, a diferencia de utilizar interfaces gráficas y controles más adaptables.

**REFERENCIAS BIBLIOGRÁFICAS**

Lehtosalo, J. (2022). *Type hints cheat sheet - mypy 1.9.0 documentation*. mypy 1.9.0 documentation. <https://mypy.readthedocs.io/en/stable/cheat_sheet_py3.html>

Python Software Foundation. (s.f.). *Typing - Support for type hints*. Python documentation. <https://docs.python.org/3/library/typing.html>