INSTITUTO SUPERIOR TECNOLÓGICO TENA
Tecnología, Innovación y Desarrollo

DESARROLLO DE SOFTWARE

Instrumento para facilitar el proceso de enseñanza-aprendizaje de la asignatura

GUÍA GENERAL DE ESTUDIO DE LA ASIGNATURA 20250020

PROGRAMACIÓN ORIENTADO A OBJETOS

Período académico
Tercero

Octubre - 2025

ING. JUAN MARCIAL ESPÍN MONTESDEOCA, MG.

GUIA GENERAL DE ESTUDIO DE LA ASIGNATURA – PROGRAMACIÓN ORIENTADA A OBJETOS

INSTITUTO SUPERIOR TECNOLÓGICO TENA

Carrera de Desarrollo de Software

ISTT DSW Primera Edición – Tena, octubre 2025

SIN ISBN

Instituto Superior Tecnológico Tena
Km. 1 1/2 Vía Tena - Archidona
Tena, Ecuador

AUTOR - REDACCIÓN Y FORMULACIÓN DE CONTENIDOS
Ing. Juan Marcial Espín Montesdeoca, Mg.

Profesor del Instituto Superior Tecnológico Tena

REVISIÓN DE PARES

Mg. Alvaro Santiago Toalombo Díaz
Mg. Henry Fabian Chango Chango
Mg. Martha Janina Duarte Mora
Mg. Danilo Alexander Zamora Núñez
Lcda. María Angélica Campoverde Encalada

Comisión de revisión técnica de guías de estudio del Instituto Superior Tecnológico Tena

APROBACIÓN

Mg. Danilo Alexander Zamora Núñez
Coordinador de Investigación, Desarrollo Tecnológico e Innovación

Impreso y hecho en Ecuador.

ÍNDICE

GUIA GENERAL DE ESTUDIO DE LA ASIGNATURA

DATOS GENERALES DE LA ASIGNATURA			
Carrera	Desarrollo de Software	**Nombre asignatura**	Programación Orientada a Objetos
Modalidad	Presencial	**Campo de Formación**	Profesional
Jornada	Matutina y Nocturna	**Unidad de Organización Curricular**	Profesional
Período académico	Segundo	**Código de la asignatura**	DSW-203
Ciclo académico	2025 IIS	**N° Total de horas de la asignatura**	192

Distribución de horas en las actividades de aprendizaje							
N° de horas Docencia	**64**	**N° de horas Aprendizaje Práctico Experimental**				**N° de horas Autónomo**	**40**
		En contacto con docente	**48**	**Autónomo**	**40**		

PRERREQUISITOS Y CORREQUISITOS			
Prerrequisitos de la asignatura		**Correquisitos de la asignatura**	
Asignatura	**Código**	**Asignatura**	**Código**

DESCRIPCIÓN DE LA ASIGNATURA

La asignatura de Programación Orientada a Objetos (POO) es un componente fundamental en la formación de cualquier estudiante de desarrollo de software. Se enfoca en enseñar los principios, técnicas y prácticas necesarios para diseñar, implementar y mantener sistemas de software utilizando el paradigma de programación orientada a objetos.

Los estudiantes adquieren conocimientos profundos sobre los conceptos fundamentales de la POO y cómo aplicarlos en la resolución de problemas reales. Esto incluye comprender la importancia de los objetos, clases, herencia, encapsulamiento, polimorfismo y abstracción, así como la relación entre ellos y su impacto en el diseño y la arquitectura de software.

OBJETIVO GENERAL

Formar a los estudiantes en el uso efectivo de este paradigma de programación y prepararlos para enfrentar los desafíos del desarrollo de software en un entorno orientado a objetos.

CONTRIBUCIÓN DE LOS RESULTADOS DE APRENDIZAJE DE LA ASIGNATURA AL PERFIL DE EGRESO DE LA CARRERA

Resultados de aprendizaje de la asignatura	Resultados de aprendizaje del perfil de egreso de la carrera	Contribución (alta – media – baja)
• Aplica conceptos, técnicas, herramientas de programación, que contribuyan con la implementación de soluciones de software. • Brinda asistencia técnica en el desarrollo de aplicaciones de software, desde el análisis del problema y la planificación del proyecto, hasta la implementación, el mantenimiento, la prueba y la documentación, y	Aplica técnicas de investigación en la búsqueda de nuevas formas de aplicación del desarrollo de software en los sectores industriales. Utiliza herramientas y tecnologías de programación para llevar a cabo tareas específicas en el campo de desarrollo de software. Brinda asistencia técnica en el desarrollo de aplicaciones de software, desde el análisis del problema y la planificación del proyecto, hasta la implementación, el	Alta Media Media

• Utiliza herramientas y tecnologías de programación para llevar a cabo tareas específicas en el campo de desarrollo de software.	mantenimiento, la prueba y la documentación. Aplica conceptos, técnicas, herramientas de programación, que contribuyan con la implementación de soluciones de software.	Media

CONTENIDOS DE LA ASIGNATURA (descripción mínima de contenidos de la asignatura)

Unidad 1

1. Principios de la programación orientada a objetos.
1.1. Generalidades
1.2. Clases y objetos
1.3. Herencia
1.4. Clases abstractas e interfaces

Unidad 2

2. Polimorfismo, manejo de errores y excepciones.
2.1 Polimorfismo
2.2 Variables y métodos estáticos
2.3 Manejo de errores y excepciones
2.4 Recursividad
2.5 Entrada y salida de datos
2.6 Ejercicios resueltos
2.7 Ejercicios propuestos

Unidad 3

3. Ficheros e hilos.
3.1 Clases fundamentales
3.2 Manejo de ficheros (archivos) con operaciones CRUD
3.3 Hilos
3.4 Creación de hilos
3.5 Multihilos
3.6 Ejercicios resueltos
3.7 Ejercicios propuestos

Unidad 4

4. Interface gráfica.
4.1 Interface gráfica
4.2 Patrón de diseño Modelo, Vistas y Controlador
4.3 Trabajar con Ficheros
4.4 Conexión a la base de datos y aplicación del CRUD
4.5 Ejercicios propuestos

ESTRATEGIAS METODOLÓGICAS Y RECURSOS DIDÁCTICOS

ESTRATEGIAS METODOLÓGICAS	HABILIDADES BLANDAS	FINALIDAD

• Activas para la enseñanza y aprendizaje	**Valores vinculados a la autonomía del sujeto:** confianza, crítica y autocrítica, honestidad, integridad	• Generar confianza/ Promover el pensamiento crítico. • Permite a los estudiantes cumplir un rol activo dentro de su formación. • Construye una sociedad participante.
• Aprendizaje y trabajo cooperativo	**Valores elementales de convivencia y civilidad:** crítica y autocrítica, tolerancia, empatía, respeto, justicia, lealtad, paciencia	• Promover un ambiente de colaboración/ trabajo en equipo/ Saber escuchar/Promover el pensamiento crítico/ fomentar el liderazgo/ adaptabilidad. • Mantener una comunicación abierta con el equipo/ tolerancia a los errores, aceptar y aprender de las críticas. • Fomentar el sentido de pertenencia
• Aprendizaje individual	**Valores vinculados a la autonomía del sujeto:** responsabilidad, honestidad, integridad, efectividad, autonomía	• Facilitar la asimilación del contenido por parte del estudiante/ Plantear preguntas para promover la comunicación efectiva /Promover el pensamiento crítico • Lectura comprensiva para fijar contenidos/ Promover el pensamiento crítico

RECURSOS DIDÁCTICOS	
MATERIALES CONVENCIONALES	*Material impreso: libros, folletos, fotocopias, periódicos, etc.*
	Tableros didácticos: pizarra
MATERIALES AUDIOVISUALES	*Imágenes fijas proyectables (fotos): diapositivas y fotografías.*
	Materiales audiovisuales (vídeo): películas y vídeos
NUEVAS TECNOLOGÍAS	*Programas informáticos: OnlyOffice, Python, Sqlit, Visual Studio Code, MySql*
	Servicios telemáticos: páginas web, plataforma EVA, correo electrónico, chats

BIBLIOGRAFÍA

Bibliografía Básica de la Asignatura:	**Físico**	**Digital**
Blasco F, (2019), *Programación orientada a objetos en Java*. España, Editorial RA-MA **ISBN:** 978-84-9964-805-7, Número de inventario biblioteca: **ISTT-DS-0093**	X	
Gervais L, (2019), *Aprender la programación orientada a objetos con el lenguaje java*. España, Editorial ENI, **ISBN:** 978-2-409-01929-6, Número de inventario biblioteca: **ISTT-DS-0152**	X	
Trejos O, Muñoz L, (2020), *Introducción a la Programación con Python*, Colombia, Editorial DGP, **ISBN:** 978-958-792-214-1, Número de inventario biblioteca: **ISTT-DS-0121**	X	
López L, (2006), *Metodología de la Programación Orientada a Objetos*, Colombia, Editorial Alfaomega, **ISBN:** 970-15-1173-5, Número de inventario biblioteca: **ISTT-DS-0036**	X	

Bibliografía de consulta de la Asignatura:	Físico	Digital
Hernandez M, Baquero L, (2022), Programación *Orientada a Objetivos en Java.* (2022 *N*úmero de inventario biblioteca: ISTT-DS-0228	X	

DESCRIPTIVA DE LAS COMPETENCIAS DE LA GUÍA DE PROGRAMACIÓN ORIENTADA A OBJETOS

UNIDAD 1: Principios de la Programación Orientada a Objetos

El estudiante comprende y aplica los fundamentos de la programación orientada a objetos para diseñar soluciones modulares, reutilizables y mantenibles que modelan situaciones del mundo real.

Competencias Específicas:

1.1 Comprensión Conceptual (Generalidades)

- Explica con claridad la diferencia entre la programación estructurada y la programación orientada a objetos
- Identifica los cuatro pilares fundamentales de la POO: encapsulación, abstracción, herencia y polimorfismo
- Reconoce situaciones del mundo real que pueden ser modeladas efectivamente usando paradigmas orientados a objetos

1.2 Diseño y Construcción (Clases y Objetos)

- Diseña clases coherentes identificando correctamente atributos y comportamientos apropiados
- Implementa objetos a partir de clases, comprendiendo la diferencia entre la plantilla (clase) y las instancias (objetos)
- Aplica los principios de encapsulación mediante modificadores de acceso (public, private, protected)
- Crea constructores apropiados para inicializar objetos de manera segura y eficiente

1.3 Reutilización y Jerarquías (Herencia)

- Diseña jerarquías de clases efectivas identificando relaciones "es un" entre entidades
- Implementa herencia simple aplicando correctamente la palabra reservada `extends` o equivalente
- Utiliza la sobreescritura de métodos para especializar comportamientos heredados
- Comprende y aplica el concepto de superclase y subclase en la construcción de soluciones

1.4 Abstracción y Contratos (Clases Abstractas e Interfaces)

- Distingue cuándo utilizar clases abstractas versus interfaces según el contexto del problema
- Define contratos de comportamiento mediante interfaces que establecen qué debe hacer una clase sin especificar el cómo
- Implementa clases abstractas que combinan comportamiento concreto con métodos abstractos
- Aplica el principio de programación hacia interfaces para crear código flexible y desacoplado

UNIDAD 2: Polimorfismo, Manejo de Errores y Excepciones

El estudiante desarrolla aplicaciones robustas y flexibles aplicando polimorfismo para crear código adaptable, implementando mecanismos de manejo de excepciones para anticipar y gestionar errores, y utilizando técnicas avanzadas de programación.

Competencias Específicas:

2.1 Flexibilidad en el Diseño (Polimorfismo)

- Comprende y aplica el polimorfismo para escribir código que trabaja con múltiples tipos de objetos de manera uniforme
- Implementa el polimorfismo de sobrecarga (mismo nombre, diferentes parámetros) y sobreescritura (redefinición en subclases)
- Utiliza referencias de tipo padre para manipular objetos de tipos hijos, aprovechando la sustitución de Liskov
- Diseña soluciones extensibles donde nuevas clases pueden agregarse sin modificar código existente

2.2 Elementos Compartidos (Variables y Métodos Estáticos)

- Identifica cuándo un atributo o comportamiento pertenece a la clase en lugar de a instancias individuales
- Implementa correctamente variables estáticas para mantener información compartida entre todos los objetos
- Crea métodos estáticos para funcionalidades que no requieren estado de instancia
- Comprende las limitaciones y mejores prácticas en el uso de elementos estáticos

2.3 Programación Defensiva (Manejo de Errores y Excepciones)

- Anticipa situaciones excepcionales que pueden interrumpir el flujo normal de ejecución
- Implementa bloques try-catch-finally para capturar y manejar excepciones apropiadamente
- Crea excepciones personalizadas para representar errores específicos del dominio de la aplicación
- Aplica estrategias de propagación de excepciones decidiendo dónde manejarlas en la jerarquía de llamadas

2.4 Pensamiento Recursivo (Recursividad)

- Identifica problemas que pueden resolverse mediante descomposición recursiva
- Diseña funciones recursivas estableciendo correctamente casos base y casos recursivos
- Comprende la diferencia entre soluciones iterativas y recursivas evaluando cuál es más apropiada
- Optimiza soluciones recursivas considerando la pila de llamadas y el rendimiento

2.5 Interacción con el Usuario (Entrada y Salida de Datos)

- Implementa mecanismos robustos para capturar datos del usuario validando la entrada
- Formatea y presenta información de salida de manera clara y profesional
- Utiliza streams de entrada/salida para diferentes fuentes de datos (teclado, consola, etc.)
- Maneja errores de E/S de manera apropiada para evitar caídas de la aplicación

2.6 y 2.7 Aplicación Práctica (Ejercicios Resueltos y Propuestos)

- Integra los conceptos de la unidad resolviendo problemas prácticos de complejidad creciente
- Analiza soluciones existentes identificando patrones y mejores prácticas
- Desarrolla soluciones propias demostrando dominio de los conceptos aprendidos

UNIDAD 3: Ficheros e Hilos

El estudiante desarrolla aplicaciones que persisten información mediante el manejo eficiente de ficheros e implementa programación concurrente con hilos para crear software responsivo y eficiente que aprovecha los recursos del sistema.

Competencias Específicas:

3.1 Fundamentos de Persistencia (Clases Fundamentales)

- Comprende la arquitectura de clases para manejo de archivos en el lenguaje (File, FileReader, FileWriter, BufferedReader, etc.)
- Identifica las diferencias entre streams de bytes y streams de caracteres eligiendo el apropiado según el contexto
- Utiliza clases de alto nivel que simplifican operaciones comunes con archivos
- Maneja rutas de archivos de manera portable considerando diferentes sistemas operativos

3.2 Operaciones con Datos Persistentes (Manejo de Ficheros con CRUD)

- **Crea** archivos y escribe información estructurada de manera eficiente
- **Lee** archivos procesando la información y cargándola en estructuras de datos apropiadas
- **Actualiza** contenido existente en archivos implementando estrategias de modificación
- **Elimina** archivos y contenido específico manejando apropiadamente los recursos del sistema
- Implementa serialización de objetos para guardar estados completos de la aplicación

3.3 Conceptos de Concurrencia (Hilos)

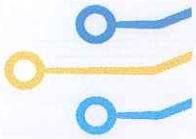
- Comprende qué es un hilo y cómo difiere de un proceso
- Identifica situaciones donde la programación concurrente mejora el rendimiento y la experiencia del usuario
- Reconoce los desafíos de la concurrencia: condiciones de carrera, deadlocks e inconsistencia de datos
- Diferencia entre concurrencia (alternancia de tareas) y paralelismo (ejecución simultánea)

3.4 Implementación Básica (Creación de Hilos)

- Crea hilos extendiendo la clase Thread o implementando la interfaz Runnable
- Inicia, pausa y controla el ciclo de vida de hilos individuales
- Comprende los estados de un hilo (nuevo, ejecutable, bloqueado, esperando, terminado)
- Implementa tareas asíncronas simples que se ejecutan en segundo plano

3.5 Programación Avanzada Concurrente (Multihilos)

- Coordina múltiples hilos que trabajan conjuntamente hacia un objetivo común



- Implementa mecanismos de sincronización (synchronized, locks) para proteger recursos compartidos
- Utiliza herramientas de comunicación entre hilos (wait, notify, notifyAll)
- Aplica patrones de concurrencia como productor-consumidor y pool de hilos
- Diseña aplicaciones responsivas separando tareas pesadas del hilo principal (UI thread)

3.6 y 3.7 Consolidación Práctica (Ejercicios Resueltos y Propuestos)

- Integra manejo de archivos con operaciones concurrentes en aplicaciones realistas
- Resuelve problemas complejos que requieren persistencia de datos y procesamiento paralelo
- Depura y optimiza código concurrente identificando y resolviendo problemas de sincronización

UNIDAD 4: Interface Gráfica

El estudiante diseña e implementa aplicaciones completas con interfaces gráficas intuitivas, aplicando patrones de diseño profesionales, integrando persistencia de datos en archivos y bases de datos, creando software de calidad profesional listo para usuarios finales.

Competencias Específicas:

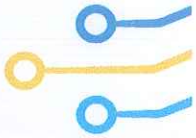
4.1 Desarrollo de Interfaces de Usuario (Interface Gráfica)

- Diseña interfaces gráficas intuitivas aplicando principios de usabilidad y experiencia de usuario
- Implementa componentes gráficos (ventanas, botones, campos de texto, tablas, menús) usando frameworks apropiados
- Gestiona eventos de usuario (clics, entradas de teclado) mediante listeners y handlers
- Crea layouts responsivos que se adaptan a diferentes tamaños de pantalla
- Aplica principios de diseño visual (contraste, alineación, jerarquía) para interfaces profesionales

4.2 Arquitectura de Software (Patrón MVC)

- Comprende la separación de responsabilidades en Modelo (datos y lógica de negocio), Vista (interfaz) y Controlador (coordinación)
- Diseña la capa Modelo con clases que representan las entidades del dominio y la lógica de la aplicación
- Implementa Vistas que presentan información sin contener lógica de negocio
- Desarrolla Controladores que median entre Vista y Modelo, gestionando el flujo de la aplicación
- Reconoce los beneficios del MVC: mantenibilidad, testabilidad y trabajo en equipo
- Aplica el patrón MVC para crear código desacoplado donde los componentes pueden evolucionar independientemente

4.3 Integración con Archivos (Trabajar con Ficheros)



- Integra la persistencia en archivos con interfaces gráficas permitiendo cargar y guardar datos
- Implementa diálogos de selección de archivos (FileChooser) para mejorar la experiencia del usuario
- Maneja operaciones de importación/exportación de datos en diferentes formatos (texto, CSV, JSON, XML)
- Proporciona feedback visual durante operaciones con archivos (barras de progreso, mensajes de estado)
- Gestiona errores de archivo mostrando mensajes comprensibles al usuario

4.4 Persistencia en Bases de Datos (Conexión a BD y CRUD)

- Establece conexiones con bases de datos relacionales usando JDBC o frameworks equivalentes
- Implementa operaciones CRUD completas desde la interfaz gráfica:
- **Create**: Formularios que insertan nuevos registros validando los datos
- **Read**: Tablas y listas que consultan y muestran información de la base de datos
- **Update**: Interfaces que permiten modificar registros existentes
- **Delete**: Funcionalidades seguras para eliminar información con confirmaciones apropiadas
- Utiliza consultas preparadas (PreparedStatements) para prevenir inyección SQL
- Maneja transacciones asegurando consistencia de datos en operaciones complejas
- Implementa patrones DAO (Data Access Object) para encapsular la lógica de acceso a datos
- Presenta errores de base de datos de manera amigable sin exponer detalles técnicos al usuario

4.5 Integración Final (Ejercicios Propuestos)

- Desarrolla aplicaciones completas que integran todos los conceptos del curso
- Crea sistemas CRUD completos con interfaz gráfica, arquitectura MVC y persistencia
- Aplica buenas prácticas de ingeniería de software: modularidad, reutilización, documentación
- Demuestra competencia profesional entregando software funcional y de calidad

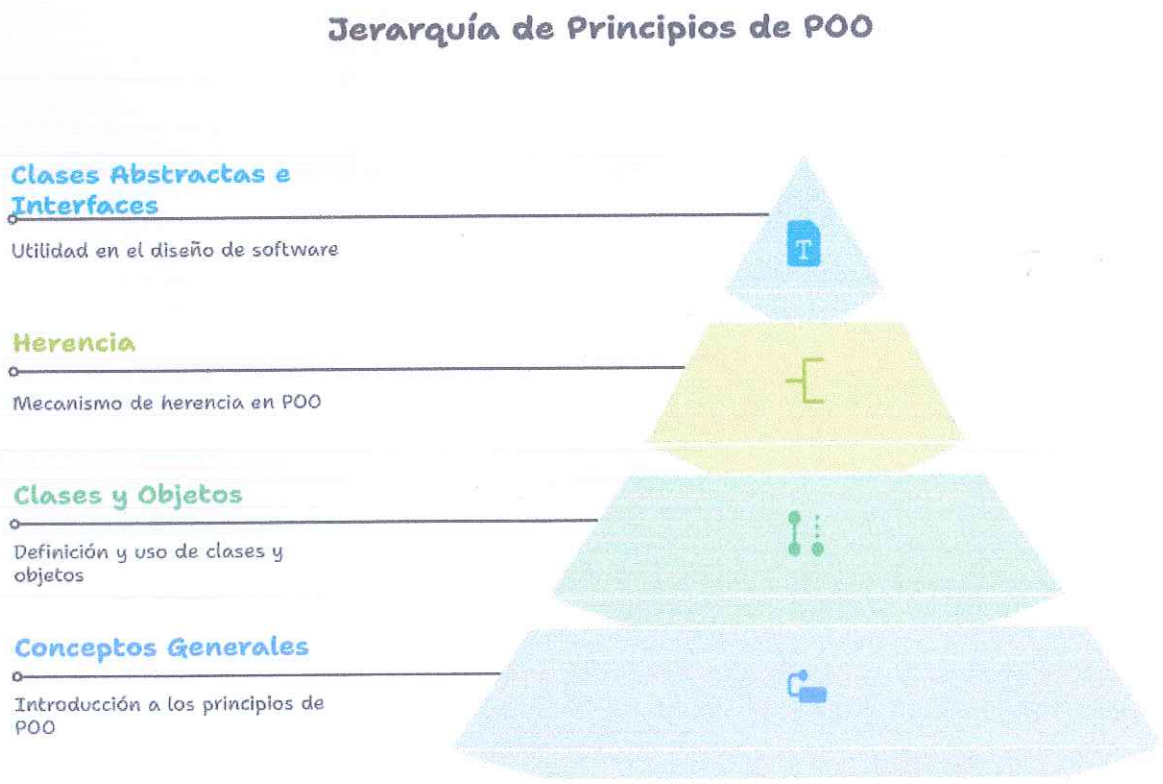
UNIDAD 1: PRINCIPIOS DE LA PROGRAMACIÓN ORIENTADA A OBJETOS.

1.1. Generalidades
1.2. Clases y objetos
1.3. Herencia
1.4. Clases abstractas e interfaces

Resultado de Aprendizaje

- Aplica conceptos, técnicas, herramientas de programación, que contribuyan con la implementación de soluciones de software.

DIAGRAMA DE APRENDIZAJE



SÍNTESIS

Esta unidad introduce los fundamentos conceptuales y prácticos de la POO. Comienza explicando por qué surgió este paradigma y cómo supera las limitaciones de la programación estructurada mediante la **encapsulación** (agrupar datos y métodos), **abstracción** (ocultar complejidad), **herencia** (reutilizar código) y **polimorfismo** (flexibilidad).

El estudiante aprende a **diseñar clases** como plantillas que definen la estructura y comportamiento de objetos, identificando qué características (atributos) y acciones (métodos) son relevantes. Se enfatiza la diferencia crucial entre clase (el molde) y objeto (la instancia concreta).

La **herencia** permite crear jerarquías de clases donde las subclases heredan y especializan características de sus superclases, modelando relaciones "es un" (un Perro "es un" Animal). Esto promueve la reutilización y organización lógica del código.

Finalmente, las **clases abstractas** e **interfaces** establecen contratos de comportamiento. Las interfaces definen "qué debe hacer" una clase sin especificar el "cómo", mientras que las clases abstractas pueden combinar implementación concreta con métodos abstractos. Estas herramientas son fundamentales para crear diseños flexibles y desacoplados.

En esencia: Esta unidad proporciona el vocabulario, los conceptos y las técnicas básicas para pensar y programar orientado a objetos, sentando las bases arquitectónicas de todo el curso.

UNIDAD 1: PRINCIPIOS DE LA PROGRAMACIÓN ORIENTADA A OBJETOS.

1.1 GENERALIDADES

¿Qué es la Programación Orientada a Objetos (POO)?

La Programación Orientada a Objetos es un paradigma de programación que organiza el código en unidades llamadas **objetos**, los cuales combinan datos (atributos) y comportamientos (métodos). Este enfoque modela el mundo real de manera más intuitiva.

Principios Fundamentales de la POO

1. Abstracción

Permite enfocarse en las características esenciales de un objeto, ocultando los detalles innecesarios.

Ejemplo del mundo real: Un automóvil

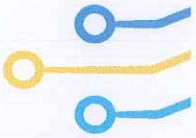
- Lo esencial: marca, modelo, color, acelerar, frenar
- Lo oculto: detalles internos del motor, sistema de inyección

2. Encapsulamiento

Agrupa datos y métodos relacionados, protegiendo la información interna del objeto.
Ejemplo: Una cuenta bancaria oculta el saldo interno y solo permite operaciones controladas (depositar, retirar).

3. Herencia

Permite crear nuevas clases basadas en clases existentes, reutilizando código.
Ejemplo: `VehículoTerrestre` → `Automóvil`, `Motocicleta`



4. Polimorfismo

Permite que objetos de diferentes clases respondan al mismo método de manera diferente.
Ejemplo: El método `hacerSonido()` en `Perro` ladra, en `Gato` maúlla.

Ventajas de la POO

- **Reutilización de código** mediante herencia
- **Modularidad** y organización clara
- **Mantenibilidad** facilitada
- **Modelado natural** del mundo real
- **Escalabilidad** en proyectos grandes

1.2 CLASES Y OBJETOS

Concepto de Clase

Una **clase** es un molde o plantilla que define las características (atributos) y comportamientos (métodos) que tendrán los objetos creados a partir de ella.

Sintaxis básica en Python:

```
class NombreClase:
    # Constructor
    def __init__(self, parametros):
        self.atributo = valor

    # Método
    def metodo(self):
        # código
        pass
```

Concepto de Objeto

Un **objeto** es una instancia específica de una clase. Es una entidad concreta con valores particulares para sus atributos.

Ejemplo Básico: Clase Persona

```
class Persona:
    # Constructor (método especial que inicializa el objeto)
    def __init__(self, nombre, edad):
        self.nombre = nombre  # Atributo público
        self.edad = edad      # Atributo público

    # Método para mostrar información
    def presentarse(self):
        return f"Hola, soy {self.nombre} y tengo {self.edad} años"

    # Método para cumplir años
    def cumplir_anios(self):
        self.edad += 1
        return f"{self.nombre} ahora tiene {self.edad} años"
```

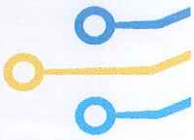
```
# Creación de objetos (instancias)
persona1 = Persona("Ana", 25)
persona2 = Persona("Carlos", 30)

# Uso de métodos
print(persona1.presentarse())  # Salida: Hola, soy Ana y tengo 25 años
print(persona2.presentarse())  # Salida: Hola, soy Carlos y tengo 30 años

# Modificar estado del objeto
print(persona1.cumplir_anios())  # Salida: Ana ahora tiene 26 años
```

Atributos de Clase vs Atributos de Instancia

```
class Estudiante:
    # Atributo de clase (compartido por todas las instancias)
    institucion = "Universidad Nacional"
    contador_estudiantes = 0

    def __init__(self, nombre, carrera):
        # Atributos de instancia (únicos para cada objeto)
        self.nombre = nombre
        self.carrera = carrera
        Estudiante.contador_estudiantes += 1

    def mostrar_info(self):
        return f"{self.nombre} estudia {self.carrera} en
{Estudiante.institucion}"

    @classmethod
    def obtener_total_estudiantes(cls):
        return f"Total de estudiantes: {cls.contador_estudiantes}"

# Crear instancias
est1 = Estudiante("María", "Ingeniería")
est2 = Estudiante("Pedro", "Medicina")

print(est1.mostrar_info())
print(Estudiante.obtener_total_estudiantes())  # Total de estudiantes: 2
```

Encapsulamiento: Atributos Públicos, Protegidos y Privados

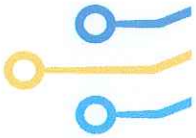
```
class CuentaBancaria:
    def __init__(self, titular, saldo_inicial):
        self.titular = titular              # Público
        self._numero_cuenta = "123456"      # Protegido (convención)
        self.__saldo = saldo_inicial        # Privado

    # Getter (método para obtener valor privado)
    def obtener_saldo(self):
        return self.__saldo

    # Setter (método para modificar valor privado con validación)
    def depositar(self, cantidad):
```



```
        if cantidad > 0:
            self.__saldo += cantidad
            return f"Depósito exitoso. Saldo actual: ${self.__saldo}"
        return "Cantidad inválida"

    def retirar(self, cantidad):
        if 0 < cantidad <= self.__saldo:
            self.__saldo -= cantidad
            return f"Retiro exitoso. Saldo actual: ${self.__saldo}"
        return "Fondos insuficientes o cantidad inválida"

# Uso
cuenta = CuentaBancaria("Juan Pérez", 1000)
print(cuenta.titular)  # Acceso directo (público)
print(cuenta.obtener_saldo())  # Acceso mediante método
print(cuenta.depositar(500))
print(cuenta.retirar(200))
# print(cuenta.__saldo)  # Esto generaría un error
```

EJERCICIO RESUELTO 1: Clase Rectángulo

```
class Rectangulo:
    """Clase que representa un rectángulo"""

    def __init__(self, base, altura):
        self.base = base
        self.altura = altura

    def calcular_area(self):
        return self.base * self.altura

    def calcular_perimetro(self):
        return 2 * (self.base + self.altura)

    def es_cuadrado(self):
        return self.base == self.altura

    def __str__(self):
        return f"Rectángulo de {self.base}x{self.altura}"

# Pruebas
rect1 = Rectangulo(5, 3)
print(rect1)
print(f"Área: {rect1.calcular_area()}")
print(f"Perímetro: {rect1.calcular_perimetro()}")
print(f"¿Es cuadrado?: {rect1.es_cuadrado()}")

rect2 = Rectangulo(4, 4)
print(f"\n{rect2}")
print(f"¿Es cuadrado?: {rect2.es_cuadrado()}")
```



EJERCICIO RESUELTO 2: Clase Producto

```
class Producto:
    """Sistema de gestión de productos"""

    iva = 0.12  # Atributo de clase (12% de IVA)

    def __init__(self, codigo, nombre, precio, stock):
        self.codigo = codigo
        self.nombre = nombre
        self.__precio = precio  # Privado
        self.__stock = stock    # Privado

    def obtener_precio(self):
        return self.__precio

    def obtener_stock(self):
        return self.__stock

    def precio_con_iva(self):
        return self.__precio * (1 + Producto.iva)

    def vender(self, cantidad):
        if cantidad <= self.__stock:
            self.__stock -= cantidad
            total = cantidad * self.precio_con_iva()
            return f"Venta exitosa. Total: ${total:.2f}. Stock restante:
{self.__stock}"
        return "Stock insuficiente"

    def reabastecer(self, cantidad):
        self.__stock += cantidad
        return f"Stock actualizado: {self.__stock} unidades"

# Uso del sistema
producto = Producto("P001", "Laptop", 800, 10)
print(f"Producto: {producto.nombre}")
print(f"Precio sin IVA: ${producto.obtener_precio()}")
print(f"Precio con IVA: ${producto.precio_con_iva():.2f}")
print(producto.vender(3))
print(producto.reabastecer(5))
```

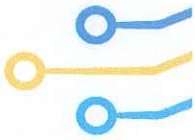
1.3 HERENCIA

Concepto

La **herencia** permite crear nuevas clases (clases hijas o derivadas) basadas en clases existentes (clases padre o base), heredando sus atributos y métodos.

Sintaxis Básica

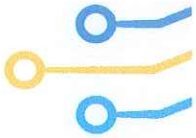
```
class ClasePadre:
    def __init__(self):
        # código
```



```
        pass

class ClaseHija(ClasePadre):
    def __init__(self):
        super().__init__()  # Llama al constructor de la clase padre
        # código adicional
```

Ejemplo: Sistema de Empleados

```
class Empleado:
    """Clase base para empleados"""

    def __init__(self, nombre, apellido, salario_base):
        self.nombre = nombre
        self.apellido = apellido
        self._salario_base = salario_base

    def nombre_completo(self):
        return f"{self.nombre} {self.apellido}"

    def calcular_salario(self):
        return self._salario_base

    def __str__(self):
      return f"{self.nombre_completo()} - Salario: ${self.calcular_salario()}"


class EmpleadoTiempoCompleto(Empleado):
    """Empleado con bonificación por tiempo completo"""

    def __init__(self, nombre, apellido, salario_base, bonificacion):
        super().__init__(nombre, apellido, salario_base)
        self.bonificacion = bonificacion

    def calcular_salario(self):
        return self._salario_base + self.bonificacion


class EmpleadoPorHoras(Empleado):
    """Empleado que cobra por horas trabajadas"""

    def __init__(self, nombre, apellido, tarifa_hora):
        super().__init__(nombre, apellido, 0)
        self.tarifa_hora = tarifa_hora
        self.horas_trabajadas = 0

    def registrar_horas(self, horas):
        self.horas_trabajadas += horas

    def calcular_salario(self):
        return self.tarifa_hora * self.horas_trabajadas
# Uso
```



```
emp1 = EmpleadoTiempoCompleto("Ana", "García", 1000, 200)
emp2 = EmpleadoPorHoras("Carlos", "López", 15)
emp2.registrar_horas(80)

print(emp1)
print(emp2)
```

Herencia Múltiple

Python permite que una clase herede de múltiples clases padre.

```
class Volador:
    def volar(self):
        return "Estoy volando"

class Nadador:
    def nadar(self):
        return "Estoy nadando"

class Pato(Volador, Nadador):
    def __init__(self, nombre):
        self.nombre = nombre

    def presentarse(self):
        return f"Soy {self.nombre}, un pato"

# Uso
donald = Pato("Donald")
print(donald.presentarse())
print(donald.volar())
print(donald.nadar())
```

EJERCICIO RESUELTO 3: Sistema de Vehículos

```
class Vehiculo:
    """Clase base para vehículos"""

    def __init__(self, marca, modelo, año):
        self.marca = marca
        self.modelo = modelo
        self.año = año
        self._kilometraje = 0

    def obtener_info(self):
        return f"{self.marca} {self.modelo} ({self.año})"

    def conducir(self, km):
        self._kilometraje += km
      return f"Has conducido {km} km. Kilometraje total: {self._kilometraje}
km"

class Auto(Vehiculo):
    def __init__(self, marca, modelo, año, num_puertas):
        super().__init__(marca, modelo, año)
```

```
        self.num_puertas = num_puertas

    def obtener_info(self):
        info_base = super().obtener_info()
        return f"{info_base} - {self.num_puertas} puertas"


class Motocicleta(Vehiculo):
    def __init__(self, marca, modelo, año, tipo):
        super().__init__(marca, modelo, año)
        self.tipo = tipo  # deportiva, crucero, touring

    def obtener_info(self):
        info_base = super().obtener_info()
        return f"{info_base} - Tipo: {self.tipo}"

    def hacer_caballito(self):
        return "¡Haciendo un caballito! 🏍️"

# Pruebas
auto = Auto("Toyota", "Corolla", 2022, 4)
moto = Motocicleta("Yamaha", "R6", 2023, "deportiva")

print(auto.obtener_info())
print(auto.conducir(150))

print(moto.obtener_info())
print(moto.conducir(80))
print(moto.hacer_caballito())
```

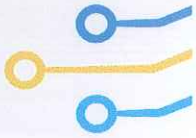
1.4 CLASES ABSTRACTAS E INTERFACES

Clases Abstractas

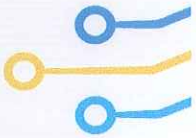
Una **clase abstracta** es una clase que no puede ser instanciada directamente y sirve como plantilla para otras clases. Contiene al menos un método abstracto que debe ser implementado por las clases hijas.

En Python se utiliza el módulo `abc` (Abstract Base Classes).

```
from abc import ABC, abstractmethod

class FiguraGeometrica(ABC):
    """Clase abstracta para figuras geométricas"""

    def __init__(self, nombre):
        self.nombre = nombre

    @abstractmethod
    def calcular_area(self):
        """Método abstracto que debe ser implementado"""
        pass
    @abstractmethod
```



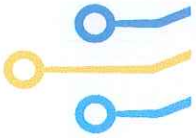
```
    def calcular_perimetro(self):
        """Método abstracto que debe ser implementado"""
        pass

    def describir(self):
        """Método concreto disponible para todas las clases hijas"""
        return f"Soy una figura geométrica: {self.nombre}"


class Circulo(FiguraGeometrica):
    def __init__(self, radio):
        super().__init__("Círculo")
        self.radio = radio

    def calcular_area(self):
        import math
        return math.pi * self.radio ** 2

    def calcular_perimetro(self):
        import math
        return 2 * math.pi * self.radio


class Triangulo(FiguraGeometrica):
    def __init__(self, base, altura, lado1, lado2, lado3):
        super().__init__("Triángulo")
        self.base = base
        self.altura = altura
        self.lado1 = lado1
        self.lado2 = lado2
        self.lado3 = lado3

    def calcular_area(self):
        return (self.base * self.altura) / 2

    def calcular_perimetro(self):
        return self.lado1 + self.lado2 + self.lado3

# Uso
# figura = FiguraGeometrica("Test")  # Esto daría ERROR (no se puede instanciar)

circulo = Circulo(5)
triangulo = Triangulo(4, 3, 3, 4, 5)

print(circulo.describir())
print(f"Área: {circulo.calcular_area():.2f}")
print(f"Perímetro: {circulo.calcular_perimetro():.2f}")

print(f"\n{triangulo.describir()}")
print(f"Área: {triangulo.calcular_area():.2f}")
print(f"Perímetro: {triangulo.calcular_perimetro():.2f}")
```



Interfaces (mediante clases abstractas)

Una **interfaz** define un contrato de métodos que las clases deben implementar, sin proporcionar implementación concreta.

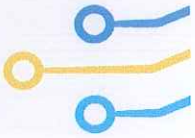
```
from abc import ABC, abstractmethod

class Reproducible(ABC):
    """Interfaz para objetos reproducibles"""

    @abstractmethod
    def reproducir(self):
        pass

    @abstractmethod
    def pausar(self):
        pass

    @abstractmethod
    def detener(self):
        pass


class ReproductorMusica(Reproducible):
    def __init__(self, cancion):
        self.cancion = cancion
        self.estado = "detenido"

    def reproducir(self):
        self.estado = "reproduciendo"
        return f"  Reproduciendo: {self.cancion}"

    def pausar(self):
        self.estado = "pausado"
        return "⏸ Música en pausa"

    def detener(self):
        self.estado = "detenido"
        return "■ Música detenida"


class ReproductorVideo(Reproducible):
    def __init__(self, video):
        self.video = video
        self.estado = "detenido"

    def reproducir(self):
        self.estado = "reproduciendo"
        return f"▶ Reproduciendo video: {self.video}"

    def pausar(self):
```



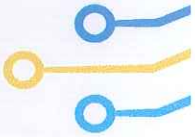
```
        self.estado = "pausado"
        return "⏸ Video en pausa"

    def detener(self):
        self.estado = "detenido"
        return "⏹ Video detenido"


# Uso polimórfico
def controlar_reproduccion(reproductor: Reproducible):
    print(reproductor.reproducir())
    print(reproductor.pausar())
    print(reproductor.detener())

musica = ReproductorMusica("Bohemian Rhapsody")
video = ReproductorVideo("Tutorial Python")

controlar_reproduccion(musica)
print()
controlar_reproduccion(video)
```

EJERCICIO RESUELTO 4: Sistema de Pagos

```
from abc import ABC, abstractmethod

class MetodoPago(ABC):
    """Interfaz para métodos de pago"""

    def __init__(self, titular):
        self.titular = titular

    @abstractmethod
    def procesar_pago(self, monto):
        pass

    @abstractmethod
    def verificar_fondos(self, monto):
        pass

class TarjetaCredito(MetodoPago):
    def __init__(self, titular, numero_tarjeta, limite):
        super().__init__(titular)
        self.numero_tarjeta = numero_tarjeta
        self.limite = limite
        self.deuda_actual = 0

    def verificar_fondos(self, monto):
        return (self.deuda_actual + monto) <= self.limite

    def procesar_pago(self, monto):
```



```
        if self.verificar_fondos(monto):
            self.deuda_actual += monto
            return f"  Pago de ${monto} procesado con tarjeta. Disponible:
${self.limite - self.deuda_actual}"
        return "X Límite de crédito insuficiente"


class PayPal(MetodoPago):
    def __init__(self, titular, email, saldo):
        super().__init__(titular)
        self.email = email
        self.saldo = saldo

    def verificar_fondos(self, monto):
        return self.saldo >= monto

    def procesar_pago(self, monto):
        if self.verificar_fondos(monto):
            self.saldo -= monto
         return f"  Pago de ${monto} procesado vía PayPal. Saldo: ${self.saldo}"
        return "X Saldo insuficiente en PayPal"

class TransferenciaBancaria(MetodoPago):
    def __init__(self, titular, numero_cuenta, saldo):
        super().__init__(titular)
        self.numero_cuenta = numero_cuenta
        self.saldo = saldo

    def verificar_fondos(self, monto):
        return self.saldo >= monto

    def procesar_pago(self, monto):
        if self.verificar_fondos(monto):
            self.saldo -= monto
         return f"  Transferencia de ${monto} realizada. Saldo: ${self.saldo}"
        return "X Fondos insuficientes"

# Sistema de procesamiento
class ProcesadorPagos:
    def realizar_pago(self, metodo: MetodoPago, monto):
        print(f"\nProcesando pago de {metodo.titular}...")
        print(metodo.procesar_pago(monto))

#.Pruebas
procesador = ProcesadorPagos()

tarjeta = TarjetaCredito("Juan Pérez", "1234-5678-9012-3456", 5000)
paypal = PayPal("María López", "maria@email.com", 1500)
transferencia = TransferenciaBancaria("Carlos Ruiz", "0012345678", 3000)

procesador.realizar_pago(tarjeta, 2000)
procesador.realizar_pago(paypal, 500)
```



```
procesador.realizar_pago(transferencia, 1000)
procesador.realizar_pago(paypal, 2000)  # Este fallará
```

EJERCICIOS PROPUESTOS

Ejercicio 1: Biblioteca Virtual
Crear un sistema de gestión de biblioteca con las siguientes clases:

Requisitos:

- Clase `ItemBiblioteca` (abstracta) con atributos: título, autor, año, código
- Métodos abstractos: `calcular_multa()`, `mostrar_info()`
- Clases hijas: `Libro` (páginas, género), `Revista` (número, mes), `DVD` (duración, formato)
- Clase `Usuario` con lista de ítems prestados
- Método para prestar y devolver ítems

Ejercicio 2: Sistema de Transporte
Diseñar un sistema de transporte urbano:
Requisitos:

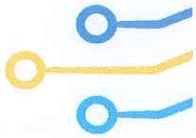
- Clase abstracta `Transporte` con atributos: capacidad, tarifa, ruta
- Método abstracto: `calcular_tarifa(distancia)`
- Clases: `Autobus`, `Metro`, `Taxi`
- Interfaz `Electrico` con método `cargar_bateria()`
- Clase `AutobusElectrico` que herede de `Autobus` e implemente `Electrico`

Ejercicio 3: Juego de Rol (RPG)
Crear un sistema de personajes para un juego:
Requisitos:

- Clase abstracta `Personaje`: nombre, nivel, vida, ataque
- Métodos abstractos: `atacar()`, `habilidad_especial()`
- Clases: `Guerrero`, `Mago`, `Arquero`
- Cada clase con atributos y habilidades únicas
- Sistema de combate entre personajes

Ejercicio 4: Tienda Online
Desarrollar un sistema de e-commerce:
Requisitos:

- Clase `Producto` con herencia a `ProductoFisico` y `ProductoDigital`
- Interfaz `Descargable` para productos digitales
- Clase `CarritoCompras` que gestione productos
- Métodos para calcular total con descuentos, impuestos y envío
- Sistema de procesamiento de órdenes



Ejercicio 5: Sistema de Notificaciones

Implementar un sistema de notificaciones:

Requisitos:

- Interfaz `Notificable` con métodos: `enviar()`, `validar()`
- Clases: `Email`, `SMS`, `NotificacionPush`
- Clase `GestorNotificaciones` que envíe notificaciones múltiples
- Implementar prioridades y logs de notificaciones

RESUMEN DE CONCEPTOS CLAVE

Concepto	Descripción	Palabra Clave Python
Clase	Plantilla para crear objetos	`class`
Objeto	Instancia de una clase	`NombreClase()`
Atributo	Variable dentro de una clase	`self.nombre`
Método	Función dentro de una clase	`def metodo(self):`
Constructor	Inicializa el objeto	`__init__`
Herencia	Reutilización de código	`class Hija(Padre):`
Clase Abstracta	No puede instanciarse	`ABC, @abstractmethod`
Polimorfismo	Mismo método, diferentes comportamientos	Sobrescritura de métodos
Encapsulamiento	Protección de datos	`_atributo, __atributo`

BUENAS PRÁCTICAS

1. **Nombres descriptivos**: Usa nombres claros para clases, métodos y atributos
2. **Responsabilidad única**: Cada clase debe tener un propósito específico
3. **Encapsulamiento**: Protege los datos con atributos privados
4. **Documentación**: Usa docstrings para documentar clases y métodos
5. **Herencia con moderación**: Solo cuando haya relación "es-un"
6. **Composición sobre herencia**: Prefiere composición cuando sea apropiado
7. **Métodos especiales**: Implementa `__str__`, `__repr__` para mejor depuración
8. **Validación**: Valida datos en constructores y setters

REFERENCIAS Y RECURSOS ADICIONALES

- Documentación oficial de Python sobre POO
- PEP 8: Guía de estilo para código Python
- Libro: "Python Object-Oriented Programming" de Dusty Phillips
- Tutorial: Real Python - Object-Oriented Programming

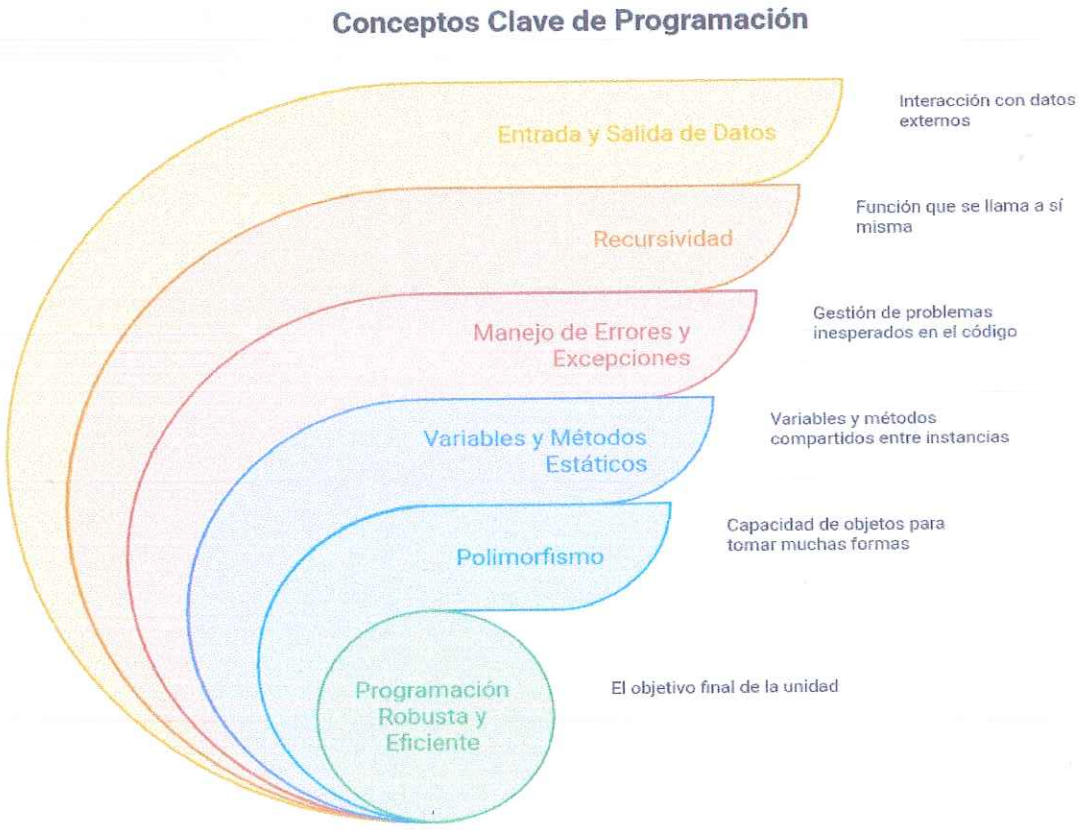
UNIDAD 2: POLIMORFISMO, MANEJO DE ERRORES Y EXCEPCIONES.

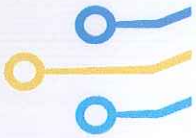
2.1 Polimorfismo
2.2 Variables y métodos estáticos
2.3 Manejo de errores y excepciones
2.4 Recursividad
2.5 Entrada y salida de datos
2.6 Ejercicios resueltos
2.7 Ejercicios propuestos

Resultado de Aprendizaje

Brinda asistencia técnica en el desarrollo de aplicaciones de software, desde el análisis del problema y la planificación del proyecto, hasta la implementación, el mantenimiento, la prueba y la documentación.

DIAGRAMA DE APRENDIZAJE





SÍNTESIS

Esta unidad profundiza en técnicas avanzadas que hacen el código más flexible y confiable. El **polimorfismo** permite que diferentes objetos respondan al mismo mensaje de maneras distintas, habilitando código que trabaja con tipos generales pero se comporta específicamente según el objeto concreto. Esto se logra mediante sobrecarga (mismo método, diferentes parámetros) y sobreescritura (redefinición en subclases).

Los **elementos estáticos** (variables y métodos) pertenecen a la clase completa en lugar de a instancias individuales, útiles para mantener información compartida o funcionalidades que no requieren estado de objeto.

El **manejo de excepciones** transforma cómo gestionamos errores: en lugar de verificar constantemente condiciones de error, el código "normal" se ejecuta en bloques `try`, y situaciones excepcionales se capturan en bloques `catch`, permitiendo respuestas apropiadas sin entorpecer la lógica principal. Las excepciones personalizadas permiten representar errores específicos del dominio de la aplicación.

La **recursividad** ofrece una técnica elegante para resolver problemas que pueden descomponerse en versiones más simples de sí mismos, fundamental en algoritmos de búsqueda, ordenamiento y procesamiento de estructuras jerárquicas.

La **entrada/salida de datos** se maneja de forma robusta, validando inputs y formateando outputs profesionalmente, cerrando el ciclo de interacción con usuarios o sistemas externos.

En esencia: Esta unidad equipa al estudiante con herramientas para crear software adaptable (polimorfismo) y resiliente (excepciones), capaz de manejar la variabilidad y los errores inherentes al mundo real.

Unidad 2: POLIMORFISMO, MANEJO DE ERRORES Y EXCEPCIONES.

2.1 POLIMORFISMO

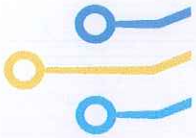
Concepto

El **polimorfismo** (del griego "muchas formas") es un principio fundamental de la Programación Orientada a Objetos que permite que objetos de diferentes clases sean tratados como objetos de una clase común. En Python, el polimorfismo se manifiesta principalmente de dos formas:

1. **Polimorfismo de sobrescritura (Override)**: Cuando una clase hija redefine un método de la clase padre
2. **Polimorfismo de interfaz**: Cuando diferentes clases implementan métodos con el mismo nombre, pero comportamientos distintos

Ventajas del Polimorfismo

- Código más flexible y reutilizable
- Facilita el mantenimiento y extensión del código
- Permite escribir código genérico que funciona con múltiples tipos



Ejemplo 1: Polimorfismo Básico

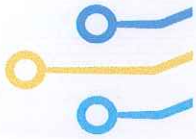
```
class Animal:
    def __init__(self, nombre):
        self.nombre = nombre

    def hacer_sonido(self):
        pass  # Método base que será sobrescrito

    def presentarse(self):
        return f"Soy {self.nombre} y hago: {self.hacer_sonido()}"

class Perro(Animal):
    def hacer_sonido(self):
        return "Guau guau"

class Gato(Animal):
    def hacer_sonido(self):
        return "Miau miau"

class Vaca(Animal):
    def hacer_sonido(self):
        return "Muuuu"

# Uso del polimorfismo
animales = [
    Perro("Firulais"),
    Gato("Michi"),
    Vaca("Lola")
]

for animal in animales:
    print(animal.presentarse())
```

Salida:

```
Soy Firulais y hago: Guau guau
Soy Michi y hago: Miau miau
Soy Lola y hago: Muuuu
```

Ejemplo 2: Polimorfismo en Formas Geométricas

```
import math

class Forma:
    def area(self):
        raise NotImplementedError("Este método debe ser implementado")
    def perimetro(self):
        raise NotImplementedError("Este método debe ser implementado")

class Circulo(Forma):
    def __init__(self, radio):
```



```
        self.radio = radio
    def area(self):
        return math.pi * self.radio ** 2

    def perimetro(self):
        return 2 * math.pi * self.radio

class Rectangulo(Forma):
    def __init__(self, base, altura):
        self.base = base
        self.altura = altura

    def area(self):
        return self.base * self.altura

    def perimetro(self):
        return 2 * (self.base + self.altura)

class Triangulo(Forma):
    def __init__(self, lado1, lado2, lado3):
        self.lado1 = lado1
        self.lado2 = lado2
        self.lado3 = lado3

    def area(self):
        # Fórmula de Herón
        s = (self.lado1 + self.lado2 + self.lado3) / 2
        return math.sqrt(s * (s - self.lado1) * (s - self.lado2) * (s -
self.lado3))

    def perimetro(self):
        return self.lado1 + self.lado2 + self.lado3

# Función polimórfica que trabaja con cualquier forma
def mostrar_informacion(forma):
    print(f"Área: {forma.area():.2f}")
    print(f"Perímetro: {forma.perimetro():.2f}")
    print("-" * 30)

# Uso
formas = [
    Circulo(5),
    Rectangulo(4, 6),
    Triangulo(3, 4, 5)
]

for forma in formas:
    mostrar_informacion(forma)
```

2.2 VARIABLES Y MÉTODOS ESTÁTICOS

Variables Estáticas (de Clase)

Las **variables estáticas** o **variables de clase** son compartidas por todas las instancias de una clase. Se definen dentro de la clase, pero fuera de cualquier método.

Métodos Estáticos

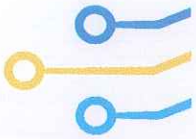
Los **métodos estáticos** pertenecen a la clase, no a las instancias. Se definen usando el decorador `@staticmethod` y no reciben `self` como primer parámetro.

Métodos de Clase

Los **métodos de clase** se definen con el decorador `@classmethod` y reciben `cls` como primer parámetro, que hace referencia a la clase misma.

Ejemplo 1: Variables y Métodos Estáticos

```
class Empleado:
    # Variable de clase (estática)
    cantidad_empleados = 0
    salario_minimo = 450.00

    def __init__(self, nombre, salario):
        self.nombre = nombre
        self.salario = salario
        Empleado.cantidad_empleados += 1

    # Método de instancia
    def obtener_info(self):
        return f"{self.nombre} - Salario: ${self.salario}"

    # Método estático
    @staticmethod
    def es_salario_valido(salario):
        return salario >= Empleado.salario_minimo

    # Método de clase
    @classmethod
    def obtener_cantidad_empleados(cls):
        return cls.cantidad_empleados

    @classmethod
    def cambiar_salario_minimo(cls, nuevo_salario):
        cls.salario_minimo = nuevo_salario
        print(f"Salario mínimo actualizado a: ${nuevo_salario}")

# Uso de métodos estáticos sin crear instancias
print(f"¿Salario de $500 es válido? {Empleado.es_salario_valido(500)}")
print(f"¿Salario de $300 es válido? {Empleado.es_salario_valido(300)}")
```



```
# Crear empleados
emp1 = Empleado("Juan Pérez", 800)
emp2 = Empleado("María García", 950)
emp3 = Empleado("Carlos López", 750)

# Acceder a variable de clase
print(f"\nTotal de empleados: {Empleado.obtener_cantidad_empleados()}")

# Cambiar valor estático
Empleado.cambiar_salario_minimo(500)
print(f"¿Salario de $480 es válido ahora? {Empleado.es_salario_valido(480)}")
```

Ejemplo 2: Contador de Objetos

```
class Producto:
    # Variables de clase
    total_productos = 0
    iva = 0.12

    def __init__(self, nombre, precio):
        self.nombre = nombre
        self.precio = precio
        self.id = Producto.total_productos + 1
        Producto.total_productos += 1

    def precio_con_iva(self):
        return self.precio * (1 + Producto.iva)

    @classmethod
    def cambiar_iva(cls, nuevo_iva):
        cls.iva = nuevo_iva

    @staticmethod
    def calcular_descuento(precio, porcentaje):
        return precio * (1 - porcentaje / 100)

    def __str__(self):
        return f"ID: {self.id} - {self.nombre}: ${self.precio:.2f}"

# Crear productos
p1 = Producto("Laptop", 850.00)
p2 = Producto("Mouse", 15.50)
p3 = Producto("Teclado", 45.00)

print(f"Total de productos creados: {Producto.total_productos}")
print(f"\n{p1}")
print(f"Precio con IVA: ${p1.precio_con_iva():.2f}")

# Usar método estático
precio_con_descuento = Producto.calcular_descuento(850, 10)
print(f"\nLaptop con 10% descuento: ${precio_con_descuento:.2f}")
```

2.3 MANEJO DE ERRORES Y EXCEPCIONES

Concepto

Las **excepciones** son eventos que ocurren durante la ejecución de un programa y que interrumpen el flujo normal de instrucciones. Python proporciona un mecanismo robusto para manejar estos errores de forma elegante.

Estructura try-except

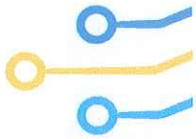
```
try:
    # Código que puede generar una excepción
    pass
except TipoDeExcepcion:
    # Código para manejar la excepción
    pass
else:
    # Se ejecuta si NO ocurre ninguna excepción
    pass
finally:
    # Se ejecuta SIEMPRE, ocurra o no una excepción
    pass
```

Excepciones Comunes en Python

Excepción	Descripción
ValueError	Valor inapropiado
TypeError	Tipo de dato incorrecto
ZeroDivisionError	División entre cero
IndexError	Índice fuera de rango
KeyError	Clave no encontrada en diccionario
FileNotFoundError	Archivo no encontrado
AttributeError	Atributo no existe

Ejemplo 1: Manejo Básico de Excepciones

```
def dividir(a, b):
    try:
        resultado = a / b
        return resultado
    except ZeroDivisionError:
        print("X Error: No se puede dividir entre cero")
        return None
    except TypeError:
        print("X Error: Los valores deben ser números")
        return None

# Pruebas
print(dividir(10, 2))       # 5.0
print(dividir(10, 0))       # Maneja división por cero
```



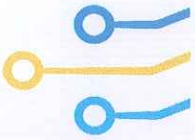
```
print(dividir(10, "2"))     # Maneja tipo incorrecto
```

Ejemplo 2: Múltiples Excepciones

```
def calcular_promedio(numeros):
    try:
        suma = sum(numeros)
        cantidad = len(numeros)
        promedio = suma / cantidad
        return promedio
    except ZeroDivisionError:
        print("X Error: La lista está vacía")
        return None
    except TypeError:
        print("X Error: La lista contiene valores no numéricos")
        return None
    finally:
        print("✓ Proceso de cálculo finalizado")

# Pruebas
print(f"Promedio: {calcular_promedio([5, 8, 10, 7])}")
print(f"Promedio: {calcular_promedio([])}")
print(f"Promedio: {calcular_promedio([1, 2, 'tres'])}")
```

Ejemplo 3: Excepciones Personalizadas

```
class SaldoInsuficienteError(Exception):
    """Excepción personalizada para saldo insuficiente"""
    def __init__(self, saldo, monto):
        self.saldo = saldo
        self.monto = monto
      self.mensaje = f"Saldo insuficiente. Saldo actual: ${saldo:.2f}, Monto
solicitado: ${monto:.2f}"
        super().__init__(self.mensaje)

class CuentaBancaria:
    def __init__(self, titular, saldo_inicial=0):
        self.titular = titular
        self.saldo = saldo_inicial

    def depositar(self, monto):
        if monto <= 0:
            raise ValueError("El monto debe ser mayor a cero")
        self.saldo += monto
        print(f"✓ Depósito exitoso. Nuevo saldo: ${self.saldo:.2f}")

    def retirar(self, monto):
        if monto <= 0:
            raise ValueError("El monto debe ser mayor a cero")
        if monto > self.saldo:
            raise SaldoInsuficienteError(self.saldo, monto)
```



```
        self.saldo -= monto
        print(f"✓ Retiro exitoso. Nuevo saldo: ${self.saldo:.2f}")

    def consultar_saldo(self):
        return f"Saldo de {self.titular}: ${self.saldo:.2f}"

# Uso
cuenta = CuentaBancaria("Ana Martínez", 500)

try:
    cuenta.depositar(200)
    cuenta.retirar(300)
    cuenta.retirar(500)  # Esto generará una excepción
except SaldoInsuficienteError as e:
    print(f"✗ {e.mensaje}")
except ValueError as e:
    print(f"✗ Error de valor: {e}")
finally:
    print(cuenta.consultar_saldo())
```

Ejemplo 4: Validación de Entrada de Usuario

```
def solicitar_edad():
    while True:
        try:
            edad = int(input("Ingrese su edad: "))
            if edad < 0 or edad > 120:
                raise ValueError("La edad debe estar entre 0 y 120")
            return edad
        except ValueError as e:
            if "invalid literal" in str(e):
                print("✗ Error: Debe ingresar un número entero")
            else:
                print(f"✗ Error: {e}")
        except KeyboardInterrupt:
            print("\n\n⚠ Operación cancelada por el usuario")
            return None

def clasificar_por_edad(edad):
    if edad is None:
        return "Operación cancelada"
    elif edad < 18:
        return "Menor de edad"
    elif edad < 65:
        return "Adulto"
    else:
        return "Adulto mayor"

# Uso (comentado para evitar input interactivo)
# edad = solicitar_edad()
# print(f"Clasificación: {clasificar_por_edad(edad)}")
```

2.4 RECURSIVIDAD

Concepto

La **recursividad** es una técnica de programación donde una función se llama a sí misma para resolver un problema dividiéndolo en subproblemas más pequeños. Toda función recursiva debe tener:

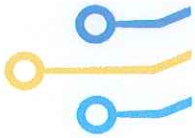
1. **Caso base**: Condición que detiene la recursión
2. **Caso recursivo**: Llamada a la función misma con parámetros modificados

Estructura de una Función Recursiva

```
def funcion_recursiva(parametros):
    # Caso base
    if condicion_de_parada:
        return valor_base

    # Caso recursivo
    return funcion_recursiva(parametros_modificados)
```

Ejemplo 1: Factorial

```
def factorial(n):
    """Calcula el factorial de n de forma recursiva"""
    # Caso base
    if n == 0 or n == 1:
        return 1
    # Caso recursivo
    return n * factorial(n - 1)

# Pruebas
print(f"Factorial de 5: {factorial(5)}")  # 120
print(f"Factorial de 0: {factorial(0)}")  # 1
print(f"Factorial de 7: {factorial(7)}")  # 5040

# Versión con validación
def factorial_seguro(n):
    try:
        if not isinstance(n, int):
            raise TypeError("El valor debe ser un entero")
        if n < 0:
            raise ValueError("El factorial no está definido para números
negativos")

        if n == 0 or n == 1:
            return 1
        return n * factorial_seguro(n - 1)
    except RecursionError:
        return "Error: Número demasiado grande (desbordamiento de pila)"

print(f"\nFactorial de 10: {factorial_seguro(10)}")
```

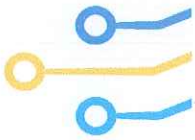


Ejemplo 2: Secuencia de Fibonacci

```
def fibonacci(n):
    """Retorna el n-ésimo número de Fibonacci"""
    if n <= 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fibonacci(n - 1) + fibonacci(n - 2)

# Mostrar los primeros 10 números de Fibonacci
print("Secuencia de Fibonacci:")
for i in range(10):
    print(f"F({i}) = {fibonacci(i)}")

# Versión optimizada con memorización
def fibonacci_memo(n, memo={}):
    """Fibonacci optimizado con memorización"""
    if n in memo:
        return memo[n]

    if n <= 0:
        return 0
    elif n == 1:
        return 1

    memo[n] = fibonacci_memo(n - 1, memo) + fibonacci_memo(n - 2, memo)
    return memo[n]

print(f"\nFibonacci(30) optimizado: {fibonacci_memo(30)}")
```

Ejemplo 3: Suma de Dígitos

```
def suma_digitos(n):
    """Suma los dígitos de un número de forma recursiva"""
    n = abs(n)  # Trabajar con valor absoluto

    # Caso base
    if n < 10:
        return n

    # Caso recursivo: último dígito + suma del resto
    return (n % 10) + suma_digitos(n // 10)

# Pruebas
print(f"Suma de dígitos de 1234: {suma_digitos(1234)}")  # 10
print(f"Suma de dígitos de 9876: {suma_digitos(9876)}")  # 30
print(f"Suma de dígitos de 555: {suma_digitos(555)}")    # 15
```



Ejemplo 4: Potencia Recursiva

```
def potencia(base, exponente):
    """Calcula base^exponente de forma recursiva"""
    # Caso base
    if exponente == 0:
        return 1

    # Caso para exponentes negativos
    if exponente < 0:
        return 1 / potencia(base, -exponente)

    # Caso recursivo
    return base * potencia(base, exponente - 1)

# Pruebas
print(f"2^5 = {potencia(2, 5)}")      # 32
print(f"3^4 = {potencia(3, 4)}")      # 81
print(f"2^-3 = {potencia(2, -3)}")    # 0.125
```

Ejemplo 5: Búsqueda Binaria Recursiva

```
def busqueda_binaria(lista, elemento, inicio=0, fin=None):
    """Busca un elemento en una lista ordenada usando búsqueda binaria
recursiva"""
    if fin is None:
        fin = len(lista) - 1

    # Caso base: elemento no encontrado
    if inicio > fin:
        return -1

    medio = (inicio + fin) // 2

    # Caso base: elemento encontrado
    if lista[medio] == elemento:
        return medio

    # Casos recursivos
    if elemento < lista[medio]:
        return busqueda_binaria(lista, elemento, inicio, medio - 1)
    else:
        return busqueda_binaria(lista, elemento, medio + 1, fin)

# Pruebas
numeros = [2, 5, 8, 12, 16, 23, 38, 45, 56, 67, 78]
print(f"Lista: {numeros}")
print(f"Buscar 23: posición {busqueda_binaria(numeros, 23)}")
print(f"Buscar 67: posición {busqueda_binaria(numeros, 67)}")
print(f"Buscar 100: posición {busqueda_binaria(numeros, 100)}")
```

2.5 ENTRADA Y SALIDA DE DATOS

Entrada de Datos (Input)

Python proporciona la función `input()` para recibir datos del usuario. Siempre retorna una cadena de texto que debe ser convertida al tipo necesario.

```
# Entrada básica
nombre = input("Ingrese su nombre: ")

# Entrada con conversión
edad = int(input("Ingrese su edad: "))
altura = float(input("Ingrese su altura en metros: "))
```

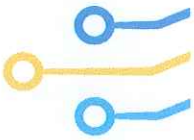
Salida de Datos (Output)

La función `print()` permite mostrar información al usuario con varias opciones de formato.

```
# Print básico
print("Hola mundo")

# Print con múltiples argumentos
print("Nombre:", nombre, "Edad:", edad)

# Print con formato f-strings (recomendado)
print(f"Hola {nombre}, tienes {edad} años")

# Print con formato
print("Pi con 2 decimales: {:.2f}".format(3.14159))
```

Ejemplo 1: Entrada de Datos con Validación

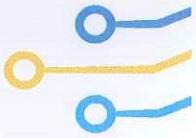
```
def solicitar_numero_entero(mensaje, minimo=None, maximo=None):
    """Solicita un número entero con validación"""
    while True:
        try:
            numero = int(input(mensaje))

            if minimo is not None and numero < minimo:
                print(f"X Error: El número debe ser mayor o igual a {minimo}")
                continue

            if maximo is not None and numero > maximo:
                print(f"X Error: El número debe ser menor o igual a {maximo}")
                continue

            return numero

        except ValueError:
            print("X Error: Debe ingresar un número entero válido")

def solicitar_numero_decimal(mensaje):
```



```
    """Solicita un número decimal con validación"""
    while True:
        try:
            numero = float(input(mensaje))
            return numero
        except ValueError:
            print("X Error: Debe ingresar un número decimal válido")

# Ejemplo de uso (comentado para evitar input interactivo)
# edad = solicitar_numero_entero("Ingrese su edad: ", minimo=0, maximo=120)
# estatura = solicitar_numero_decimal("Ingrese su estatura en metros: ")
```

Ejemplo 2: Manejo de Archivos - Escritura

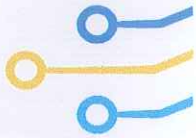
```
class GestorArchivos:
    @staticmethod
    def escribir_archivo_texto(nombre_archivo, contenido):
        """Escribe contenido en un archivo de texto"""
        try:
            with open(nombre_archivo, 'w', encoding='utf-8') as archivo:
                archivo.write(contenido)
            print(f"✓ Archivo '{nombre_archivo}' creado exitosamente")
            return True
        except IOError as e:
            print(f"X Error al escribir el archivo: {e}")
            return False

    @staticmethod
    def agregar_a_archivo(nombre_archivo, contenido):
        """Agrega contenido al final de un archivo"""
        try:
            with open(nombre_archivo, 'a', encoding='utf-8') as archivo:
                archivo.write(contenido + '\n')
            print(f"✓ Contenido agregado a '{nombre_archivo}'")
            return True
        except IOError as e:
            print(f"X Error al agregar al archivo: {e}")
            return False

    @staticmethod
    def leer_archivo_texto(nombre_archivo):
        """Lee el contenido de un archivo de texto"""
        try:
            with open(nombre_archivo, 'r', encoding='utf-8') as archivo:
                contenido = archivo.read()
            return contenido
        except FileNotFoundError:
            print(f"X Error: El archivo '{nombre_archivo}' no existe")
            return None
        except IOError as e:
            print(f"X Error al leer el archivo: {e}")
            return None
# Ejemplo de uso
```



```
contenido = """Este es un archivo de ejemplo.
Contiene varias líneas de texto.
Python facilita el manejo de archivos."""

GestorArchivos.escribir_archivo_texto("ejemplo.txt", contenido)
GestorArchivos.agregar_a_archivo("ejemplo.txt", "Línea adicional")
print("\nContenido del archivo:")
print(GestorArchivos.leer_archivo_texto("ejemplo.txt"))
```

Ejemplo 3: Entrada/Salida con JSON

```
import json

class GestorJSON:
    @staticmethod
    def guardar_json(nombre_archivo, datos):
        """Guarda datos en formato JSON"""
        try:
            with open(nombre_archivo, 'w', encoding='utf-8') as archivo:
                json.dump(datos, archivo, indent=4, ensure_ascii=False)
            print(f"✓ Datos guardados en '{nombre_archivo}'")
            return True
        except Exception as e:
            print(f"X Error al guardar JSON: {e}")
            return False

    @staticmethod
    def cargar_json(nombre_archivo):
        """Carga datos desde un archivo JSON"""
        try:
            with open(nombre_archivo, 'r', encoding='utf-8') as archivo:
                datos = json.load(archivo)
            return datos
        except FileNotFoundError:
            print(f"X Error: El archivo '{nombre_archivo}' no existe")
            return None
        except json.JSONDecodeError:
            print(f"X Error: El archivo no contiene JSON válido")
            return None

# Ejemplo de uso
estudiantes = {
    "estudiantes": [
        {"nombre": "Juan Pérez", "edad": 20, "promedio": 8.5},
        {"nombre": "María García", "edad": 22, "promedio": 9.2},
        {"nombre": "Carlos López", "edad": 21, "promedio": 7.8}
    ]
}


GestorJSON.guardar_json("estudiantes.json", estudiantes)
datos_cargados = GestorJSON.cargar_json("estudiantes.json")
```



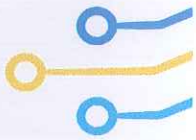
```
if datos_cargados:
    print("\nEstudiantes cargados:")
    for est in datos_cargados["estudiantes"]:
        print(f"- {est['nombre']}: Promedio {est['promedio']}")
```

2.6 EJERCICIOS RESUELTOS

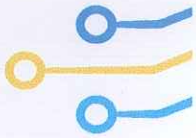
Ejercicio 1: Sistema de Biblioteca con Polimorfismo

Enunciado: Crear un sistema de biblioteca que maneje diferentes tipos de materiales (libros, revistas, DVDs) con polimorfismo.

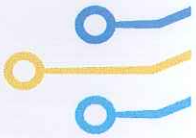
```
from datetime import datetime, timedelta

class MaterialBiblioteca:
    def __init__(self, titulo, codigo):
        self.titulo = titulo
        self.codigo = codigo
        self.prestado = False
        self.fecha_prestamo = None

    def prestar(self):
        if self.prestado:
            return False
        self.prestado = True
        self.fecha_prestamo = datetime.now()
        return True

    def devolver(self):
        self.prestado = False
        self.fecha_prestamo = None

    def dias_prestamo(self):
        """Método que será sobrescrito"""
        raise NotImplementedError

    def calcular_multa(self):
        if not self.prestado or not self.fecha_prestamo:
            return 0

        dias_transcurridos = (datetime.now() - self.fecha_prestamo).days
        dias_permitidos = self.dias_prestamo()

        if dias_transcurridos > dias_permitidos:
            dias_atraso = dias_transcurridos - dias_permitidos
            return dias_atraso * 0.50  # $0.50 por día de atraso
        return 0


    def __str__(self):
        estado = "Prestado" if self.prestado else "Disponible"
```



```
        return f"{self.codigo} - {self.titulo} [{estado}]"

class Libro(MaterialBiblioteca):
    def __init__(self, titulo, codigo, autor, isbn):
        super().__init__(titulo, codigo)
        self.autor = autor
        self.isbn = isbn

    def dias_prestamo(self):
        return 14  # 14 días para libros

    def __str__(self):
        return f"📚 LIBRO: {super().__str__()} - Autor: {self.autor}"

class Revista(MaterialBiblioteca):
    def __init__(self, titulo, codigo, numero_edicion):
        super().__init__(titulo, codigo)
        self.numero_edicion = numero_edicion

    def dias_prestamo(self):
        return 7  # 7 días para revistas

    def __str__(self):
      return f"  REVISTA: {super().__str__()} - Edición: {self.numero_edicion}"

class DVD(MaterialBiblioteca):
    def __init__(self, titulo, codigo, duracion):
        super().__init__(titulo, codigo)
        self.duracion = duracion  # en minutos

    def dias_prestamo(self):
        return 3  # 3 días para DVDs

    def __str__(self):
        return f"💿 DVD: {super().__str__()} - Duración: {self.duracion} min"

class Biblioteca:
    def __init__(self, nombre):
        self.nombre = nombre
        self.materiales = []

    def agregar_material(self, material):
        self.materiales.append(material)
        print(f"✓ Material agregado: {material.titulo}")

    def prestar_material(self, codigo):
        for material in self.materiales:
            if material.codigo == codigo:
                if material.prestar():
                    print(f"✓ Material prestado exitosamente")
                print(f"  Debe devolverlo en {material.dias_prestamo()} días")
```



```
                    return True
                else:
                    print(f"X El material ya está prestado")
                    return False
        print(f"X Material con código {codigo} no encontrado")
        return False

    def devolver_material(self, codigo):
        for material in self.materiales:
            if material.codigo == codigo:
                multa = material.calcular_multa()
                material.devolver()
                print(f"✓ Material devuelto exitosamente")
                if multa > 0:
                    print(f"⚠ Multa por atraso: ${multa:.2f}")
                return True
        print(f"X Material con código {codigo} no encontrado")
        return False

    def listar_materiales(self):
        print(f"\n=== MATERIALES DE {self.nombre} ===")
        if not self.materiales:
            print("No hay materiales registrados")
        else:
            for material in self.materiales:
                print(material)
                if material.prestado:
                    multa = material.calcular_multa()
                    if multa > 0:
                        print(f"  ⚠ Multa acumulada: ${multa:.2f}")

# Ejemplo de uso
biblioteca = Biblioteca("Biblioteca Central")

# Agregar materiales
biblioteca.agregar_material(Libro("Cien Años de Soledad", "L001", "Gabriel
García Márquez", "978-0307474728"))
biblioteca.agregar_material(Libro("1984", "L002", "George Orwell", "978-
0451524935"))
biblioteca.agregar_material(Revista("National Geographic", "R001", "Marzo
2025"))
biblioteca.agregar_material(DVD("El Padrino", "D001", 175))

# Listar materiales
biblioteca.listar_materiales()

# Realizar préstamos
print("\n--- PRÉSTAMOS ---")
biblioteca.prestar_material("L001")
biblioteca.prestar_material("D001")

# Listar después de préstamos
```

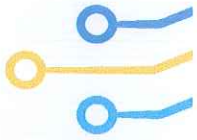


```
biblioteca.listar_materiales()

# Devolver
print("\n--- DEVOLUCIONES ---")
biblioteca.devolver_material("L001")
```

Ejercicio 2: Calculadora con Manejo de Excepciones

Enunciado: Crear una calculadora robusta que maneje diferentes tipos de excepciones y operaciones.

```
class ErrorDivisionCero(Exception):
    """Excepción personalizada para división por cero"""
    pass

class ErrorOperacionInvalida(Exception):
    """Excepción para operaciones no soportadas"""
    pass

class Calculadora:
    def __init__(self):
        self.historial = []

    def realizar_operacion(self, operacion, num1, num2):
        """Realiza una operación matemática con manejo de excepciones"""
        try:
            # Validar tipos de datos
         if not isinstance(num1, (int, float)) or not isinstance(num2, (int,
float)):
                raise TypeError("Los operandos deben ser números")

            # Realizar operación
            if operacion == '+':
                resultado = num1 + num2
            elif operacion == '-':
                resultado = num1 - num2
            elif operacion == '*':
                resultado = num1 * num2
            elif operacion == '/':
                if num2 == 0:
                    raise ErrorDivisionCero("No se puede dividir entre cero")
                resultado = num1 / num2
            elif operacion == '//':
                if num2 == 0:
                    raise ErrorDivisionCero("No se puede dividir entre cero")
                resultado = num1 // num2
            elif operacion == '%':
                if num2 == 0:
                raise ErrorDivisionCero("No se puede obtener módulo de cero")
                resultado = num1 % num2
            elif operacion == '**':
                resultado = num1 ** num2
            else:
```



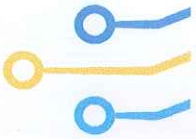
```
                raise ErrorOperacionInvalida(f"Operación '{operacion}' no
soportada")

            # Guardar en historial
            operacion_str = f"{num1} {operacion} {num2} = {resultado}"
            self.historial.append(operacion_str)

            return resultado

        except ErrorDivisionCero as e:
            print(f"X Error de división: {e}")
            return None
        except ErrorOperacionInvalida as e:
            print(f"X {e}")
            return None
        except TypeError as e:
            print(f"X Error de tipo: {e}")
            return None
        except OverflowError:
            print(f"X Error: El resultado es demasiado grande")
            return None
        except Exception as e:
            print(f"X Error inesperado: {e}")
            return None

    def calcular_expresion(self, expresion):
        """Evalúa una expresión matemática completa"""
        try:
            # Advertencia de seguridad: eval puede ser peligroso
            # En producción, usar un parser seguro
            resultado = eval(expresion)
            self.historial.append(f"{expresion} = {resultado}")
            return resultado
        except ZeroDivisionError:
            print("X Error: División entre cero en la expresión")
            return None
        except SyntaxError:
            print("X Error: Expresión matemática inválida")
            return None
        except Exception as e:
            print(f"X Error al evaluar expresión: {e}")
            return None

    def mostrar_historial(self):
        """Muestra el historial de operaciones"""
        print("\n=== HISTORIAL DE OPERACIONES ===")
        if not self.historial:
            print("No hay operaciones en el historial")
        else:
            for i, operacion in enumerate(self.historial, 1):
                print(f"{i}. {operacion}")
```

```
    def limpiar_historial(self):
        """Limpia el historial de operaciones"""
        self.historial.clear()
        print("✓ Historial limpiado")

# Ejemplo de uso
calc = Calculadora()

print("=== PRUEBAS DE CALCULADORA ===\n")

# Operaciones básicas
print(f"5 + 3 = {calc.realizar_operacion('+', 5, 3)}")
print(f"10 - 4 = {calc.realizar_operacion('-', 10, 4)}")
print(f"6 * 7 = {calc.realizar_operacion('*', 6, 7)}")
print(f"15 / 3 = {calc.realizar_operacion('/', 15, 3)}")

# Manejo de errores
print(f"\n--- Pruebas de manejo de errores ---")
print(f"10 / 0 = {calc.realizar_operacion('/', 10, 0)}")
print(f"5 + 'texto' = {calc.realizar_operacion('+', 5, 'texto')}")
print(f"5 & 3 = {calc.realizar_operacion('&', 5, 3)}")

# Operaciones avanzadas
print(f"\n--- Operaciones avanzadas ---")
print(f"2 ** 10 = {calc.realizar_operacion('**', 2, 10)}")
print(f"17 % 5 = {calc.realizar_operacion('%', 17, 5)}")
print(f"17 // 5 = {calc.realizar_operacion('//', 17, 5)}")

# Expresiones completas
print(f"\n--- Expresiones ---")
print(f"(5 + 3) * 2 = {calc.calcular_expresion('(5 + 3) * 2')}")
print(f"2 ** 8 / 4 = {calc.calcular_expresion('2 ** 8 / 4')}")

# Mostrar historial
calc.mostrar_historial()
```

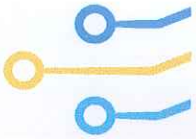
Ejercicio 3: Sistema de Archivos de Estudiantes

Enunciado: Crear un sistema que gestione información de estudiantes usando archivos JSON con manejo completo de excepciones.

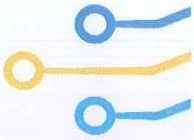
```
import json
from datetime import datetime

class Estudiante:
    def __init__(self, id_estudiante, nombre, apellido, edad, carrera):
        self.id_estudiante = id_estudiante
        self.nombre = nombre
        self.apellido = apellido
        self.edad = edad
        self.carrera = carrera
        self.materias = []
```



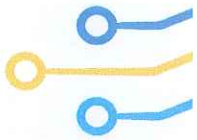
```
    def agregar_materia(self, nombre_materia, calificacion):
        if calificacion < 0 or calificacion > 10:
            raise ValueError("La calificación debe estar entre 0 y 10")

        self.materias.append({
            "nombre": nombre_materia,
            "calificacion": calificacion
        })

    def calcular_promedio(self):
        if not self.materias:
            return 0
        total = sum(materia["calificacion"] for materia in self.materias)
        return total / len(self.materias)

    def to_dict(self):
        """Convierte el objeto a diccionario para JSON"""
        return {
            "id_estudiante": self.id_estudiante,
            "nombre": self.nombre,
            "apellido": self.apellido,
            "edad": self.edad,
            "carrera": self.carrera,
            "materias": self.materias
        }

    @staticmethod
    def from_dict(data):
        """Crea un objeto Estudiante desde un diccionario"""
        estudiante = Estudiante(
            data["id_estudiante"],
            data["nombre"],
            data["apellido"],
            data["edad"],
            data["carrera"]
        )
        estudiante.materias = data.get("materias", [])
        return estudiante

    def __str__(self):
        promedio = self.calcular_promedio()
        return   f"[{self.id_estudiante}]   {self.nombre}   {self.apellido}   -
{self.carrera} (Promedio: {promedio:.2f})"

class SistemaEstudiantes:
    def __init__(self, archivo="estudiantes.json"):
        self.archivo = archivo
        self.estudiantes = []
        self.cargar_datos()

    def cargar_datos(self):
```



```
        """Carga los estudiantes desde el archivo JSON"""
        try:
            with open(self.archivo, 'r', encoding='utf-8') as f:
                datos = json.load(f)
            self.estudiantes = [Estudiante.from_dict(est) for est in datos]
            print(f"✓ Datos cargados exitosamente ({len(self.estudiantes)}
estudiantes)")
        except FileNotFoundError:
            print(f"⚠ Archivo '{self.archivo}' no encontrado. Se creará uno
nuevo.")
            self.estudiantes = []
        except json.JSONDecodeError:
         print(f"✗ Error: El archivo '{self.archivo}' no contiene JSON válido")
            self.estudiantes = []
        except Exception as e:
            print(f"✗ Error inesperado al cargar datos: {e}")
            self.estudiantes = []

    def guardar_datos(self):
        """Guarda los estudiantes en el archivo JSON"""
        try:
            datos = [est.to_dict() for est in self.estudiantes]
            with open(self.archivo, 'w', encoding='utf-8') as f:
                json.dump(datos, f, indent=4, ensure_ascii=False)
            print(f"✓ Datos guardados exitosamente en '{self.archivo}'")
            return True
        except IOError as e:
            print(f"✗ Error al guardar datos: {e}")
            return False
        except Exception as e:
            print(f"✗ Error inesperado al guardar: {e}")
            return False

    def agregar_estudiante(self, estudiante):
        """Agrega un nuevo estudiante"""
        try:
            # Verificar que no exista el ID
            if any(est.id_estudiante == estudiante.id_estudiante for est in
self.estudiantes):
                raise ValueError(f"Ya existe un estudiante con ID
{estudiante.id_estudiante}")

            self.estudiantes.append(estudiante)
            print(f"✓ Estudiante {estudiante.nombre} {estudiante.apellido}
agregado")
            return True
        except ValueError as e:
            print(f"✗ Error: {e}")
            return False

    def buscar_estudiante(self, id_estudiante):
        """Busca un estudiante por ID"""
```



```
        for estudiante in self.estudiantes:
            if estudiante.id_estudiante == id_estudiante:
                return estudiante
        return None

    def eliminar_estudiante(self, id_estudiante):
        """Elimina un estudiante por ID"""
        estudiante = self.buscar_estudiante(id_estudiante)
        if estudiante:
            self.estudiantes.remove(estudiante)
            print(f"✓ Estudiante {estudiante.nombre} {estudiante.apellido}
eliminado")
            return True
        else:
            print(f"X No se encontró estudiante con ID {id_estudiante}")
            return False

    def listar_estudiantes(self):
        """Lista todos los estudiantes"""
        print("\n=== LISTA DE ESTUDIANTES ===")
        if not self.estudiantes:
            print("No hay estudiantes registrados")
        else:
            for est in self.estudiantes:
                print(est)
                if est.materias:
                    print("  Materias:")
                    for materia in est.materias:
                        print(f"                    -    {materia['nombre']}:
{materia['calificacion']}")

    def generar_reporte(self):
        """Genera un reporte estadístico"""
        if not self.estudiantes:
            print("No hay estudiantes para generar reporte")
            return

        print("\n=== REPORTE ESTADÍSTICO ===")
        print(f"Total de estudiantes: {len(self.estudiantes)}")

        promedios = [est.calcular_promedio() for est in self.estudiantes if
est.materias]

        if promedios:
            promedio_general = sum(promedios) / len(promedios)
            mejor_promedio = max(promedios)
            peor_promedio = min(promedios)

            print(f"Promedio general: {promedio_general:.2f}")
            print(f"Mejor promedio: {mejor_promedio:.2f}")
            print(f"Promedio más bajo: {peor_promedio:.2f}")
```

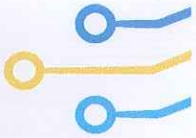


```
            # Encontrar estudiantes con mejor y peor promedio
            mejor_estudiante = next(est for est in self.estudiantes if
est.calcular_promedio() == mejor_promedio)
            print(f"\nMejor estudiante: {mejor_estudiante.nombre}
{mejor_estudiante.apellido}")

# Ejemplo de uso
sistema = SistemaEstudiantes("estudiantes_sistema.json")

# Agregar estudiantes
est1 = Estudiante("E001", "Juan", "Pérez", 20, "Ingeniería en Sistemas")
est1.agregar_materia("Programación", 9.5)
est1.agregar_materia("Matemáticas", 8.7)
est1.agregar_materia("Física", 9.0)

est2 = Estudiante("E002", "María", "García", 22, "Ingeniería Civil")
est2.agregar_materia("Cálculo", 9.8)
est2.agregar_materia("Estructuras", 9.2)

est3 = Estudiante("E003", "Carlos", "López", 21, "Medicina")
est3.agregar_materia("Anatomía", 8.5)
est3.agregar_materia("Fisiología", 8.9)

sistema.agregar_estudiante(est1)
sistema.agregar_estudiante(est2)
sistema.agregar_estudiante(est3)

# Listar estudiantes
sistema.listar_estudiantes()

# Generar reporte
sistema.generar_reporte()

# Guardar datos
sistema.guardar_datos()
```

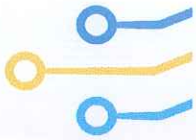
Ejercicio 4: Torre de Hanoi (Recursividad)

Enunciado: Resolver el problema clásico de la Torre de Hanoi usando recursividad.

```
class TorreHanoi:
    def __init__(self, num_discos):
        self.num_discos = num_discos
        self.movimientos = []
        self.contador = 0

    def resolver(self, n, origen, destino, auxiliar):
        """
        Resuelve la Torre de Hanoi recursivamente
        n: número de discos
        origen: torre de origen
        destino: torre de destino
        auxiliar: torre auxiliar
        """
```



```
        if n == 1:
            self.contador += 1
            movimiento = f"Movimiento {self.contador}: Mover disco 1 de {origen}
a {destino}"
            self.movimientos.append(movimiento)
            print(movimiento)
            return

        # Mover n-1 discos de origen a auxiliar usando destino
        self.resolver(n - 1, origen, auxiliar, destino)

        # Mover el disco más grande de origen a destino
        self.contador += 1
        movimiento = f"Movimiento {self.contador}: Mover disco {n} de {origen}
a {destino}"
        self.movimientos.append(movimiento)
        print(movimiento)

        # Mover n-1 discos de auxiliar a destino usando origen
        self.resolver(n - 1, auxiliar, destino, origen)

    def ejecutar(self):
        """Ejecuta la solución y muestra estadísticas"""
        print(f"\n=== TORRE DE HANOI CON {self.num_discos} DISCOS ===\n")
        self.resolver(self.num_discos, 'A', 'C', 'B')

        print(f"\n=== ESTADÍSTICAS ===")
        print(f"Total de movimientos: {self.contador}")
        print(f"Movimientos mínimos teóricos: {2**self.num_discos - 1}")

    def guardar_solucion(self, archivo="hanoi_solucion.txt"):
        """Guarda la solución en un archivo"""
        try:
            with open(archivo, 'w', encoding='utf-8') as f:
                f.write(f"SOLUCIÓN TORRE DE HANOI - {self.num_discos} DISCOS\n")
                f.write("=" * 50 + "\n\n")
                for movimiento in self.movimientos:
                    f.write(movimiento + "\n")
                f.write(f"\nTotal de movimientos: {self.contador}\n")
            print(f"✓ Solución guardada en '{archivo}'")
        except IOError as e:
            print(f"X Error al guardar solución: {e}")

# Resolver con diferentes números de discos
print("Ejemplo con 3 discos:")
hanoi3 = TorreHanoi(3)
hanoi3.ejecutar()

print("\n" + "="*60)
print("\nEjemplo con 4 discos:")
hanoi4 = TorreHanoi(4)
hanoi4.ejecutar()
```

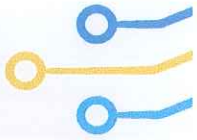


```
hanoi4.guardar_solucion()
```

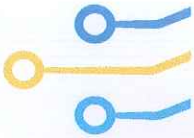
Ejercicio 5: Sistema de Facturación Completo

Enunciado: Crear un sistema de facturación que integre todos los conceptos: polimorfismo, excepciones, entrada/salida y recursividad.

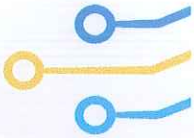
```
from datetime import datetime
import json

class ProductoNoEncontradoError(Exception):
    """Excepción cuando un producto no existe"""
    pass

class StockInsuficienteError(Exception):
    """Excepción cuando no hay suficiente stock"""
    def __init__(self, producto, solicitado, disponible):
        self.producto = producto
        self.solicitado = solicitado
        self.disponible = disponible
        super().__init__(f"Stock insuficiente de '{producto}': solicitado
{solicitado}, disponible {disponible}")

class Producto:
    contador_id = 0

    def __init__(self, nombre, precio, stock):
        Producto.contador_id += 1
        self.id = Producto.contador_id
        self.nombre = nombre
        self.precio = precio
        self.stock = stock

    def calcular_precio(self, cantidad):
        """Método base para calcular precio"""
        return self.precio * cantidad

    def verificar_stock(self, cantidad):
        return self.stock >= cantidad

    def reducir_stock(self, cantidad):
        if not self.verificar_stock(cantidad):
            raise StockInsuficienteError(self.nombre, cantidad, self.stock)
        self.stock -= cantidad

    def to_dict(self):
        return {
            "id": self.id,
            "tipo": self.__class__.__name__,
            "nombre": self.nombre,
            "precio": self.precio,
            "stock": self.stock
        }
```



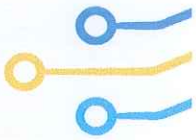
```
    def __str__(self):
        return f"[{self.id}] {self.nombre} - ${self.precio:.2f} (Stock:
{self.stock})"

class ProductoConDescuento(Producto):
    def __init__(self, nombre, precio, stock, descuento):
        super().__init__(nombre, precio, stock)
        self.descuento = descuento  # Porcentaje

    def calcular_precio(self, cantidad):
        precio_base = super().calcular_precio(cantidad)
        descuento_aplicado = precio_base * (self.descuento / 100)
        return precio_base - descuento_aplicado

    def to_dict(self):
        data = super().to_dict()
        data["descuento"] = self.descuento
        return data

    def __str__(self):
        return f"{super().__str__()} - Descuento: {self.descuento}%"

class ProductoPorMayor(Producto):
   def __init__(self, nombre, precio, stock, cantidad_minima, descuento_mayor):
        super().__init__(nombre, precio, stock)
        self.cantidad_minima = cantidad_minima
        self.descuento_mayor = descuento_mayor

    def calcular_precio(self, cantidad):
        precio_base = super().calcular_precio(cantidad)
        if cantidad >= self.cantidad_minima:
            descuento = precio_base * (self.descuento_mayor / 100)
            return precio_base - descuento
        return precio_base

    def to_dict(self):
        data = super().to_dict()
        data["cantidad_minima"] = self.cantidad_minima
        data["descuento_mayor"] = self.descuento_mayor
        return data

    def __str__(self):
        return f"{super().__str__()} - Compra mínima: {self.cantidad_minima}
({self.descuento_mayor}% desc.)"

class Factura:
    contador_factura = 0

    def __init__(self, cliente):
        Factura.contador_factura += 1
        self.numero = Factura.contador_factura
```



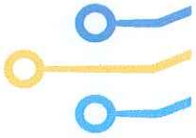
```
        self.cliente = cliente
        self.fecha = datetime.now()
        self.items = []
        self.subtotal = 0
        self.iva = 0.12
        self.total = 0

    def agregar_item(self, producto, cantidad):
        try:
            if cantidad <= 0:
                raise ValueError("La cantidad debe ser mayor a cero")

            producto.reducir_stock(cantidad)
            precio = producto.calcular_precio(cantidad)

            item = {
                "producto": producto.nombre,
                "cantidad": cantidad,
                "precio_unitario": producto.precio,
                "precio_total": precio
            }
            self.items.append(item)
            self.calcular_totales()
            print(f"✓ {cantidad} x {producto.nombre} agregado a la factura")

        except StockInsuficienteError as e:
            print(f"X {e}")
            raise
        except ValueError as e:
            print(f"X Error: {e}")
            raise

    def calcular_totales(self):
        """Calcula subtotal, IVA y total de forma recursiva"""
        def sumar_items(items, indice=0):
            if indice >= len(items):
                return 0
            return items[indice]["precio_total"] + sumar_items(items, indice
+ 1)

        self.subtotal = sumar_items(self.items)
        self.total = self.subtotal * (1 + self.iva)

    def generar_texto(self):
        """Genera el texto de la factura"""
        texto = "\n" + "=" * 60 + "\n"
        texto += f"{'FACTURA':^60}\n"
        texto += "=" * 60 + "\n"
        texto += f"Número: {self.numero:06d}\n"
        texto += f"Cliente: {self.cliente}\n"
        texto += f"Fecha: {self.fecha.strftime('%d/%m/%Y %H:%M:%S')}\n"
        texto += "=" * 60 + "\n\n"
```



```
        texto += f"{'PRODUCTO':<30} {'CANT':>6} {'P.UNIT':>10} {'TOTAL':>10}\n"
        texto += "-" * 60 + "\n"

        for item in self.items:
            texto += f"{item['producto']:<30} {item['cantidad']:>6} "
            texto             +=             f"${item['precio_unitario']:>9.2f}
${item['precio_total']:>9.2f}\n"

        texto += "-" * 60 + "\n"
        texto += f"{'SUBTOTAL:':<48} ${self.subtotal:>9.2f}\n"
        texto += f"{'IVA (12%):':<48} ${self.subtotal * self.iva:>9.2f}\n"
        texto += f"{'TOTAL:':<48} ${self.total:>9.2f}\n"
        texto += "=" * 60 + "\n"

        return texto

    def imprimir(self):
        """Imprime la factura en consola"""
        print(self.generar_texto())

    def guardar(self, archivo=None):
        """Guarda la factura en un archivo"""
        if archivo is None:
            archivo = f"factura_{self.numero:06d}.txt"

        try:
            with open(archivo, 'w', encoding='utf-8') as f:
                f.write(self.generar_texto())
            print(f"✓ Factura guardada en '{archivo}'")
            return True
        except IOError as e:
            print(f"✗ Error al guardar factura: {e}")
            return False

class SistemaFacturacion:
    def __init__(self):
        self.productos = []
        self.facturas = []

    def agregar_producto(self, producto):
        self.productos.append(producto)
        print(f"✓ Producto agregado: {producto.nombre}")

    def listar_productos(self):
        print("\n=== CATÁLOGO DE PRODUCTOS ===")
        if not self.productos:
            print("No hay productos disponibles")
        else:
            for producto in self.productos:
                print(producto)

    def buscar_producto(self, id_producto):
```



```
        for producto in self.productos:
            if producto.id == id_producto:
                return producto
        raise ProductoNoEncontradoError(f"Producto con ID {id_producto} no
encontrado")

    def crear_factura(self, cliente, items):
        """
        Crea una factura con los items especificados
        items: lista de tuplas (id_producto, cantidad)
        """
        try:
            factura = Factura(cliente)

            for id_producto, cantidad in items:
                producto = self.buscar_producto(id_producto)
                factura.agregar_item(producto, cantidad)

            self.facturas.append(factura)
            factura.imprimir()
            factura.guardar()

            return factura

      except (ProductoNoEncontradoError, StockInsuficienteError, ValueError)
as e:
            print(f"X Error al crear factura: {e}")
            return None

    def generar_reporte_ventas(self):
        """Genera un reporte de ventas"""
        if not self.facturas:
            print("No hay facturas registradas")
            return

        print("\n=== REPORTE DE VENTAS ===")
        total_ventas = sum(f.total for f in self.facturas)
        print(f"Total de facturas: {len(self.facturas)}")
        print(f"Ventas totales: ${total_ventas:.2f}")
      print(f"Promedio por factura: ${total_ventas / len(self.facturas):.2f}")

# ===  DE USO DEL SISTEMA ===
print("=== SISTEMA DE FACTURACIÓN ===\n")

# `Crear sistema
sistema = SistemaFacturacion()

# Agregar productos
sistema.agregar_producto(Producto("Laptop Dell", 850.00, 10))
sistema.agregar_producto(ProductoConDescuento("Mouse Logitech", 25.00, 50,
15))
sistema.agregar_producto(ProductoPorMayor("Cable HDMI", 8.50, 100, 10, 20))
```



```
sistema.agregar_producto(Producto("Teclado Mecánico", 75.00, 15))
sistema.agregar_producto(ProductoConDescuento("Monitor LG 24''", 180.00, 8,
10))

# Listar productos
sistema.listar_productos()

# Crear facturas
print("\n--- CREANDO FACTURA 1 ---")
sistema.crear_factura("Juan Pérez", [
    (1, 1),    # 1 Laptop
    (2, 2),    # 2 Mouse
    (3, 5)     # 5 Cables
])

print("\n--- CREANDO FACTURA 2 ---")
sistema.crear_factura("María García", [
    (4, 2),    # 2 Teclados
    (5, 1),    # 1 Monitor
    (3, 15)    # 15 Cables (aplica descuento por mayor)
])

# Generar reporte
sistema.generar_reporte_ventas()

# Mostrar stock actualizado
print("\n--- STOCK DESPUÉS DE LAS VENTAS ---")
sistema.listar_productos()
```

2.7 EJERCICIOS PROPUESTOS

Ejercicio 1: Sistema de Transporte Público
Dificultad: Media
Desarrollar un sistema de transporte público que incluya:

- Clase base `Vehiculo` con atributos: capacidad, pasajeros_actuales, tarifa_base
- Clases derivadas: `Autobus`, `Metro`, `Taxi`
- Implementar polimorfismo para calcular tarifas (el taxi cobra por kilómetro, el bus y metro por zona)
- Manejo de excepciones para: capacidad excedida, tarifa inválida
- Métodos estáticos para calcular estadísticas de la flota
- Guardar y cargar información de viajes en archivos JSON

Requerimientos técnicos:
python

```
# Estructura sugerida
class Vehiculo:
    total_vehiculos = 0
```

```
    def calcular_tarifa(self):
        pass  # Implementar en clases derivadas

    def abordar_pasajeros(self, cantidad):
        pass  # Validar capacidad
```

Ejercicio 2: Calculadora de Expresiones con Árbol de Sintaxis
Dificultad: Alta

Crear una calculadora que evalúe expresiones matemáticas usando recursividad:

- Parsear expresiones con paréntesis: `"((2 + 3) * (4 - 1)) / 5"`
- Implementar recursividad para evaluar subexpresiones
- Manejo de excepciones: paréntesis desbalanceados, operadores inválidos, división por cero
- Generar un árbol de sintaxis abstracta (ASA) de la expresión
- Guardar el historial de evaluaciones en archivo

Desafío adicional: Implementar funciones matemáticas como `sqrt()`, `pow()`, `sin()`, `cos()`

Ejercicio 3: Sistema de Gestión de Hospital
Dificultad: Alta

Desarrollar un sistema completo de hospital que integre:

- Clases: `Persona` (base), `Paciente`, `Doctor`, `Enfermero`
- Polimorfismo en métodos: `calcular_costo_atencion()`, `generar_reporte()`
- Sistema de citas médicas con manejo de excepciones
- Recursividad para buscar especialistas en una jerarquía de departamentos
- Entrada/salida: registros médicos en JSON, generación de reportes en TXT
- Variables estáticas: contador de pacientes, estadísticas globales

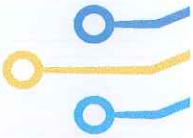
Funcionalidades requeridas:

- Agendar citas (validar disponibilidad)
- Registrar historiales médicos
- Calcular costos según tipo de atención y seguro
- Generar reportes diarios/mensuales
- Sistema de búsqueda recursiva de doctores disponibles

Ejercicio 4: Gestor de Árbol Genealógico
Dificultad: Media-Alta

Crear un sistema que maneje árboles genealógicos usando recursividad:

- Clase `Persona` con atributos: nombre, fecha_nacimiento, padre, madre, hijos
- Métodos recursivos:



- `contar_descendientes()`: cuenta todos los descendientes
- `buscar_ancestro(nombre)`: busca un ancestro en el árbol
- `calcular_generacion(persona_objetivo)`: calcula distancia generacional
- `obtener_arbol_completo()`: retorna estructura completa
- Manejo de excepciones: personas duplicadas, relaciones inválidas
- Entrada/salida: cargar familia desde JSON, exportar árbol a formato visual
- Métodos de clase para estadísticas: promedio de hijos, generaciones totales

Ejercicio 5: Sistema de Reservas de Hotel
Dificultad: Media

Implementar un sistema de reservas con:

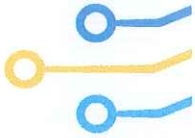
- Polimorfismo: diferentes tipos de habitaciones (Simple, Doble, Suite) con precios distintos
- Excepción personalizada: `HabitacionNoDisponibleError`
- Validación de fechas y cálculo de noches
- Variables estáticas: habitaciones totales, ocupación actual
- Métodos estáticos: calcular ocupación promedio, ingresos totales
- Persistencia: guardar reservas en JSON, generar facturas en TXT
- Sistema de descuentos: por temporada, por estancia prolongada

Ejercicio 6: Analizador de Texto Recursivo
Dificultad: Media
Crear un analizador que procese texto usando recursividad:

- Contar palabras, vocales, consonantes de forma recursiva
- Identificar palíndromos recursivamente
- Invertir texto usando recursión
- Validar estructura de paréntesis/corchetes/llaves balanceados (recursivo)
- Manejo de excepciones: archivos no encontrados, codificación incorrecta
- Entrada: leer archivos de texto
- Salida: generar reportes estadísticos en JSON y TXT

Funciones recursivas requeridas:
python

```
def es_palindromo(texto, inicio, fin):
    pass

def balancear_simbolos(texto, indice, pila):
    pass

def contar_vocales_recursivo(texto, indice):
    pass
```



Ejercicio 7: Sistema de Inventario Multi-bodega
Dificultad: Alta

Desarrollar un sistema de inventario con:

- Polimorfismo: diferentes categorías de productos (Perecedero, NoPerecedero, Electrónico)
- Cada producto calcula su valor de manera diferente (con depreciación, vencimiento, etc.)
- Excepciones: `StockInsuficienteError`, `ProductoVencidoError`, `InventarioLlenoError`
- Recursividad: buscar producto en múltiples bodegas (estructura de árbol)
- Variables estáticas: valor total del inventario, productos totales
- Entrada/salida: importar catálogo desde CSV, exportar reportes a JSON/Excel
- Sistema de alertas: stock mínimo, productos próximos a vencer

Ejercicio 8: Simulador de Red Social
Dificultad: Alta

Crear un simulador básico de red social:

- Clase `Usuario` con métodos polimórficos para diferentes tipos (Regular, Premium, Admin)
- Recursividad: encontrar conexiones de N grados, sugerir amigos
- Excepciones: `UsuarioYaExisteError`, `ConexionInvalidaError`
- Métodos estáticos: estadísticas de la red (usuarios activos, conexiones promedio)
- Persistencia: guardar grafo social en JSON
- Funcionalidades:
- Agregar/eliminar usuarios
- Crear conexiones (amistad)
- Buscar camino más corto entre usuarios (recursivo/BFS)
- Generar feed de publicaciones

Ejercicio 9: Calculadora de Impuestos Corporativos
Dificultad: Media

Sistema para calcular impuestos con:

- Polimorfismo: diferentes tipos de empresas (Pequeña, Mediana, Grande) con tasas distintas
- Excepciones: `IngresoInvalidoError`, `DeduccionExcesivaError`
- Recursividad: calcular deducciones en cascada
- Variables de clase: tasas impositivas actuales, umbral de ingresos
- Métodos estáticos: calcular proyecciones, comparar escenarios
- Entrada/salida: leer estados financieros (JSON), generar declaraciones (PDF/TXT)

Ejercicio 10: Sistema de Encriptación de Archivos
Dificultad: Media-Alta

Desarrollar un sistema de encriptación:

- Polimorfismo: diferentes algoritmos de cifrado (César, Sustitución, XOR)
- Recursividad: encriptar estructuras de datos anidadas
- Excepciones: `ClaveInvalidaError`, `ArchivoCorruptoError`
- Métodos estáticos: generar claves, validar fortaleza de contraseña
- Funcionalidades:
- Encriptar/desencriptar archivos de texto
- Validar integridad con checksums
- Guardar archivos encriptados con metadata
- Sistema de recuperación con manejo de errores robusto

RECURSOS ADICIONALES Y BUENAS PRÁCTICAS

Principios SOLID aplicados a los ejercicios

1. **Single Responsibility (Responsabilidad Única)**
   - Cada clase debe tener una única responsabilidad
   - Ejemplo: separar lógica de negocio de persistencia
2. **Open/Closed (Abierto/Cerrado)**
   - Usar polimorfismo para extender funcionalidad sin modificar código existente
3. **Liskov Substitution (Sustitución de Liskov)**
   - Las clases derivadas deben poder sustituir a las clases base
4. **Interface Segregation (Segregación de Interfaces)**
   - Crear interfaces específicas en lugar de generales
5. **Dependency Inversion (Inversión de Dependencias)**
   - Depender de abstracciones, no de implementaciones concretas

6. Consejos para resolver los ejercicios

Planificación:

- Diseña las clases y sus relaciones antes de codificar
- Define claramente qué excepciones necesitas
- Identifica dónde aplicar recursividad

Implementación:

- Comienza con casos simples y ve agregando complejidad
- Prueba cada método individualmente
- Usa docstrings para documentar tu código

Manejo de Errores:

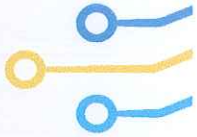
- Nunca uses `try-except` vacío
- Captura excepciones específicas, no genéricas
- Proporciona mensajes de error claros

Recursividad:

- Siempre define caso base claro
- Verifica que la recursión converge
- Considera límites de profundidad

Persistencia:

- Valida datos antes de guardar



- Usa `with` para manejo de archivos
- Implementa respaldos automáticos

Herramientas recomendadas

```
python
# Para testing
import unittest
import pytest

# Para validación de datos
from typing import List, Dict, Optional
from dataclasses import dataclass

# Para fechas
from datetime import datetime, timedelta

# Para datos estructurados
import json
import csv
```

EVALUACIÓN Y RÚBRICA

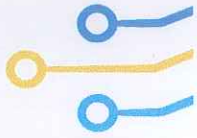
Criterios de evaluación para ejercicios

Criterio	Excelente (4)	Bueno (3)	Regular (2)	Insuficiente (1)
Polimorfismo	Implementación correcta y elegante	Funcional con pequeños errores	Implementación básica	No implementado
Excepciones	Manejo robusto y específico	Manejo básico funcional	Manejo incompleto	Sin manejo
Recursividad	Eficiente con caso base claro	Funcional pero ineficiente	Implementación confusa	No implementado
Entrada/Salida	Persistencia completa con validación	Funcional básica	Implementación parcial	No implementado
Código limpio	Bien estructurado y documentado	Organizado, pero poco documentado	Desorganizado	Ilegible
Funcionalidad	Todas las características funcionan	Mayoría funciona	Funcionalidad básica	No funciona

Checklist de autoevaluación

Antes de entregar un ejercicio, verifica:

- ¿Implementé todas las clases requeridas?
- ¿Usé polimorfismo de manera efectiva?
- ¿Manejé todas las excepciones posibles?
- ¿Mis funciones recursivas tienen caso base?
- ¿Implementé entrada/salida de datos correctamente?
- ¿Usé métodos y variables estáticas donde corresponde?
- ¿Mi código está documentado?
- ¿Probé todos los casos de uso?
- ¿El código sigue las convenciones de Python (PEP 8)?



- ¿Implementé validaciones de datos?

CONCLUSIONES DE LA UNIDAD

Conceptos clave aprendidos

1. **Polimorfismo:** Permite escribir código flexible que trabaja con múltiples tipos de objetos
2. **Variables y Métodos Estáticos:** Útiles para datos compartidos y operaciones independientes de instancias
3. **Manejo de Excepciones:** Esencial para código robusto y mantenible
4. **Recursividad:** Técnica poderosa para problemas que se pueden dividir en subproblemas
5. **Entrada/Salida:** Fundamental para persistencia y comunicación con el usuario

Aplicaciones en el mundo real

- **Polimorfismo:** Sistemas de pago (diferentes métodos de pago), frameworks de UI (diferentes componentes)
- **Excepciones:** Aplicaciones empresariales robustas, sistemas críticos
- **Recursividad:** Algoritmos de búsqueda, procesamiento de árboles, análisis sintáctico
- **Persistencia:** Cualquier aplicación que necesite guardar datos entre sesiones

Próximos pasos

- Practicar con proyectos personales
- Estudiar patrones de diseño (Strategy, Factory, Observer)
- Explorar programación funcional en Python
- Aprender sobre testing unitario y TDD
- Investigar optimización de código recursivo (memorización, programación dinámica)

¡Fin de la Unidad 2!

"El código es como el humor. Cuando tienes que explicarlo, es malo." - Cory House

Recuerda: La práctica constante es la clave para dominar estos conceptos. ¡Sigue programando!

UNIDAD 3: FICHEROS E HILOS.

3.1 Clases fundamentales
3.2 Manejo de ficheros (archivos) con operaciones CRUD
3.3 Hilos
3.4 Creación de hilos
3.5 Multihilos
3.6 Ejercicios resueltos
3.7 Ejercicios propuestos

Resultado de Aprendizaje

Utiliza herramientas y tecnologías de programación para llevar a cabo tareas específicas en el campo de desarrollo de software.

DIAGRAMA DE APRENDIZAJE

SINTESIS

Esta unidad aborda dos capacidades críticas del software profesional: **persistencia** y **concurrencia.**

El manejo de **ficheros** permite que los datos sobrevivan al cierre del programa. El estudiante domina las clases fundamentales para leer y escribir archivos, comprendiendo la diferencia entre streams de bytes (datos binarios) y caracteres (texto). Implementa operaciones **CRUD completas**: crear archivos, leer información, actualizar contenido existente y eliminar datos, junto con técnicas de serialización que permiten guardar estados completos de objetos.

Los **hilos** introducen la programación concurrente, donde múltiples flujos de ejecución ocurren simultáneamente dentro de una aplicación. Esto es esencial para interfaces responsivas (el programa responde al usuario mientras procesa tareas pesadas en segundo plano) y para aprovechar procesadores multinúcleo.

El estudiante aprende a crear hilos individuales y progresa hacia **multihilos**, donde varios hilos coordinan su trabajo. Esto introduce desafíos como condiciones de carrera (múltiples hilos accediendo datos simultáneamente) y deadlocks (hilos esperándose mutuamente indefinidamente), requiriendo mecanismos de **sincronización** para proteger recursos compartidos y coordinar actividades.

En esencia: Esta unidad transforma programas de "úsalo y piérdelo" en aplicaciones que guardan información persistentemente, y de secuenciales en concurrentes, aprovechando plenamente las capacidades del hardware moderno.

UNIDAD 3: FICHEROS E HILOS.

3.1 CLASES FUNDAMENTALES

3.1.1 La Clase open() y el Objeto File

En Python, los ficheros se manejan mediante la función built-in `open()` que retorna un objeto de tipo `file`. Este es el punto de entrada para todas las operaciones con archivos.

Sintaxis:

```
fichero = open(nombre_archivo, modo)
```

Parámetros principales:

- `nombre_archivo`: Ruta del archivo (string)
- `modo`: Tipo de acceso (string)

Modos de apertura:

Modo	Descripción
'r'	Lectura (por defecto). Genera error si no existe
'w'	Escritura. Crea el archivo o lo sobrescribe
'a'	Añadir contenido al final del archivo
'x'	Creación exclusiva. Genera error si existe
'b'	Modo binario (se combina: 'rb', 'wb')
't'	Modo texto (por defecto, 'rt', 'wt')
'+'	Lectura y escritura simultánea ('r+', 'w+')

3.1.2 Métodos Principales de la Clase File

Lectura:

- `read(n)`: Lee n caracteres (todo si no se especifica)
- `readline()`: Lee una línea completa
- `readlines()`: Lee todas las líneas como lista

Escritura:

- `write(texto)`: Escribe texto en el archivo
- `writelines(lista)`: Escribe una lista de líneas

Gestión:

- `close()`: Cierra el archivo
- `seek(posición)`: Mueve el cursor a una posición
- `tell()`: Retorna la posición actual del cursor
- `flush()`: Vacía el buffer de escritura

3.1.3 Context Manager: La sentencia `with`

La forma más segura y recomendada de trabajar con archivos es usar la sentencia `with`, que cierra automáticamente el archivo.

```
# Forma tradicional (menos segura)
f = open('archivo.txt', 'r')
contenido = f.read()
f.close()

# Forma recomendada con context manager
with open('archivo.txt', 'r') as f:
    contenido = f.read()
```

3.2 MANEJO DE FICHEROS: OPERACIONES CRUD

3.2.1 CREATE (Crear/Escribir)

Ejemplo 1: Crear un archivo con contenido básico

```
# Escritura simple
with open('estudiantes.txt', 'w') as f:
    f.write("Juan Pérez\n")
    f.write("María González\n")
    f.write("Carlos López\n")

# Escritura con writelines
lineas = ["Ana Martínez\n", "Pedro Sánchez\n", "Laura Rodríguez\n"]
with open('estudiantes2.txt', 'w') as f:
    f.writelines(lineas)
```

Ejemplo 2: Crear archivo JSON

```
import json

# Crear archivo JSON
datos = {
    "estudiantes": [
        {"id": 1, "nombre": "Juan", "calificacion": 8.5},
        {"id": 2, "nombre": "María", "calificacion": 9.2},
        {"id": 3, "nombre": "Carlos", "calificacion": 7.8}
    ]
}

with open('datos.json', 'w', encoding='utf-8') as f:
    json.dump(datos, f, ensure_ascii=False, indent=4)
```

3.2.2 READ (Leer)

Ejemplo 1: Lectura completa del archivo

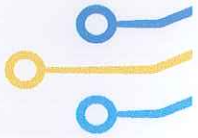
```
# Leer todo el contenido
with open('estudiantes.txt', 'r') as f:
    contenido = f.read()
    print(contenido)

# Leer línea por línea
with open('estudiantes.txt', 'r') as f:
    for linea in f:
        print(linea.strip())  # strip() elimina espacios y saltos de línea

# Leer todas las líneas como lista
with open('estudiantes.txt', 'r') as f:
    lineas = f.readlines()
    print(f"Total de líneas: {len(lineas)}")
    for i, linea in enumerate(lineas, 1):
        print(f"{i}. {linea.strip()}")
```

Ejemplo 2: Lectura selectiva de líneas

```
# Leer solo primeras n líneas
def leer_primeras_n_lineas(nombre_archivo, n):
    with open(nombre_archivo, 'r') as f:
        lineas = []
        for i, linea in enumerate(f):
            if i < n:
                lineas.append(linea.strip())
            else:
                break
    return lineas

primeras = leer_primeras_n_lineas('estudiantes.txt', 2)
print(primeras)
```

Ejemplo 3: Leer archivo JSON

```
import json
```



```
with open('datos.json', 'r', encoding='utf-8') as f:
    datos = json.load(f)

for estudiante in datos['estudiantes']:
    print(f"ID: {estudiante['id']}, Nombre: {estudiante['nombre']}, Nota:
{estudiante['calificacion']}")
```

3.2.3 UPDATE (Actualizar)

Ejemplo 1: Actualizar línea específica

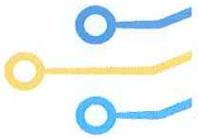
```
def actualizar_linea(nombre_archivo, numero_linea, nuevo_contenido):
    """Actualiza una línea específica del archivo"""
    with open(nombre_archivo, 'r') as f:
        lineas = f.readlines()

    # Validar que el número de línea existe
    if 0 <= numero_linea < len(lineas):
        lineas[numero_linea] = nuevo_contenido + '\n'

    with open(nombre_archivo, 'w') as f:
        f.writelines(lineas)

# Uso
actualizar_linea('estudiantes.txt', 0, 'Juan Carlos Pérez')
```

Ejemplo 2: Añadir contenido al final

```
def agregar_linea(nombre_archivo, contenido):
    """Añade una nueva línea al final del archivo"""
    with open(nombre_archivo, 'a') as f:
        f.write(contenido + '\n')

# Uso
agregar_linea('estudiantes.txt', 'Sofia Díaz')
```

Ejemplo 3: Actualizar archivo JSON

```
import json

def actualizar_estudiante(nombre_archivo, id_estudiante, nueva_calificacion):
    """Actualiza la calificación de un estudiante"""
    with open(nombre_archivo, 'r', encoding='utf-8') as f:
        datos = json.load(f)

    # Buscar y actualizar
    for estudiante in datos['estudiantes']:
        if estudiante['id'] == id_estudiante:
            estudiante['calificacion'] = nueva_calificacion
            break

    with open(nombre_archivo, 'w', encoding='utf-8') as f:
        json.dump(datos, f, ensure_ascii=False, indent=4)

# Uso
```



```
actualizar_estudiante('datos.json', 1, 9.0)
```

3.2.4 DELETE (Eliminar)

Ejemplo 1: Eliminar línea específica

```
def eliminar_linea(nombre_archivo, numero_linea):
    """Elimina una línea específica del archivo"""
    with open(nombre_archivo, 'r') as f:
        lineas = f.readlines()

    if 0 <= numero_linea < len(lineas):
        del lineas[numero_linea]

    with open(nombre_archivo, 'w') as f:
        f.writelines(lineas)

# Uso
eliminar_linea('estudiantes.txt', 1)
```

Ejemplo 2: Eliminar registro de JSON

```
def eliminar_estudiante(nombre_archivo, id_estudiante):
    """Elimina un estudiante por su ID"""
    with open(nombre_archivo, 'r', encoding='utf-8') as f:
        datos = json.load(f)

    # Filtrar el estudiante
    datos['estudiantes'] = [e for e in datos['estudiantes'] if e['id'] !=
id_estudiante]

    with open(nombre_archivo, 'w', encoding='utf-8') as f:
        json.dump(datos, f, ensure_ascii=False, indent=4)

# Uso
eliminar_estudiante('datos.json', 2)
```

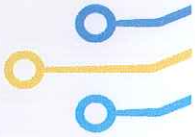
Ejemplo 3: Limpiar archivo

```
def limpiar_archivo(nombre_archivo):
    """Vacía completamente el archivo"""
    with open(nombre_archivo, 'w') as f:
        pass  # Solo abre y cierra, borrando contenido

# Uso
# limpiar_archivo('archivo_temporal.txt')
```

3.3 HILOS (THREADS)

3.3.1 Conceptos Fundamentales

Un **hilo (thread)** es la unidad de ejecución más pequeña dentro de un proceso. Los hilos comparten el mismo espacio de memoria pero tienen su propio contexto de ejecución. Esto permite que múltiples tareas se ejecuten aparentemente simultáneamente.



Ventajas:

- Ejecución aparentemente simultánea de múltiples tareas
- Mejor utilización de recursos del procesador
- Aplicaciones más responsivas
- Compartición de memoria entre hilos

Desventajas:

- Mayor complejidad del código
- Posibles problemas de sincronización
- Dificultad en la depuración
- Condiciones de carrera

3.3.2 El Módulo `threading`

```
import threading
```

Principales clases y funciones:

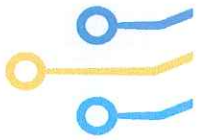
Elemento	Descripción
`Thread()`	Clase para crear hilos
`Thread.start()`	Inicia la ejecución del hilo
`Thread.join()`	Espera a que termine el hilo
`Thread.is_alive()`	Verifica si el hilo está activo
`current_thread()`	Obtiene el hilo actual
`active_count()`	Número de hilos activos
`Lock()`	Mecanismo para sincronización

3.4 CREACIÓN DE HILOS

3.4.1 Método 1: Usando la clase Thread directamente

Ejemplo 1: Hilo simple

```
import threading
import time

def tarea_simple(nombre):
    """Función que será ejecutada por el hilo"""
    for i in range(3):
        print(f"{nombre} - Iteración {i + 1}")
        time.sleep(1)
    print(f"{nombre} - Terminó")

# Crear un hilo
hilo1 = threading.Thread(target=tarea_simple, args=("Hilo 1",))
```



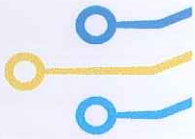
```
# Iniciar el hilo
hilo1.start()

# Esperar a que termine
hilo1.join()

print("Programa terminado")
```

Ejemplo 2: Múltiples hilos

```
import threading
import time

def trabajador(id_trabajador, tareas):
    for tarea in range(tareas):
        print(f"Trabajador {id_trabajador} realizando tarea {tarea + 1}")
        time.sleep(0.5)
    print(f"Trabajador {id_trabajador} finalizó")

# Crear múltiples hilos
hilos = []
for i in range(3):
    hilo = threading.Thread(target=trabajador, args=(i + 1, 5))
    hilos.append(hilo)
    hilo.start()

# Esperar a que todos terminen
for hilo in hilos:
    hilo.join()

print("Todos los trabajadores terminaron")
```

3.4.2 Método 2: Subclasificar Thread

```
import threading
import time

class MiHilo(threading.Thread):
    def __init__(self, nombre, duracion):
        super().__init__()
        self.nombre = nombre
        self.duracion = duracion

    def run(self):
        """Método que se ejecuta cuando inicia el hilo"""
        for i in range(self.duracion):
            print(f"{self.nombre} - Paso {i + 1}/{self.duracion}")
            time.sleep(1)
        print(f"{self.nombre} - Completado")

# Crear instancias de hilos personalizados
hilo1 = MiHilo("Proceso A", 3)
hilo2 = MiHilo("Proceso B", 4)
```



```
# Iniciar hilos
hilo1.start()
hilo2.start()

# Esperar a que terminen
hilo1.join()
hilo2.join()

print("Ejecución finalizada")
```

3.4.3 Obtener información de hilos

```
import threading
import time

def tarea():
    hilo_actual = threading.current_thread()
    print(f"Hilo: {hilo_actual.name}")
    print(f"ID del hilo: {hilo_actual.ident}")
    time.sleep(2)

# Crear y ejecutar hilos
hilo = threading.Thread(target=tarea, name="MiHilo")
hilo.start()

print(f"Hilos activos: {threading.active_count()}")
print(f"¿Hilo vivo?: {hilo.is_alive()}")

hilo.join()
print(f"¿Hilo vivo después de join?: {hilo.is_alive()}")
```

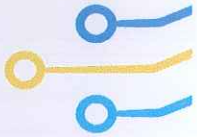
3.5 MULTIHILOS

3.5.1 Sincronización con Lock

Cuando múltiples hilos acceden a recursos compartidos, pueden ocurrir condiciones de carrera. El `Lock` (candado) previene esto.

Ejemplo 1: Problema sin sincronización

```
import threading
import time

contador = 0

def incrementar_sin_lock():
    global contador
    for _ in range(100000):
        contador += 1

# Sin sincronización - Resultado inconsistente
hilo1 = threading.Thread(target=incrementar_sin_lock)
```



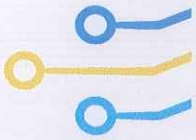
```
hilo2 = threading.Thread(target=incrementar_sin_lock)

hilo1.start()
hilo2.start()

hilo1.join()
hilo2.join()

print(f"Contador sin lock: {contador}")  # Menor a 200000
```

Ejemplo 2: Solución con Lock

```
import threading

contador = 0
lock = threading.Lock()

def incrementar_con_lock():
    global contador
    for _ in range(100000):
        with lock:  # Context manager
            contador += 1

# Con sincronización - Resultado consistente
hilo1 = threading.Thread(target=incrementar_con_lock)
hilo2 = threading.Thread(target=incrementar_con_lock)

hilo1.start()
hilo2.start()

hilo1.join()
hilo2.join()

print(f"Contador con lock: {contador}")  # Siempre 200000
```

3.5.2 Operaciones con archivos compartidos

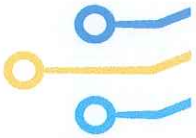
```
import threading
import time

class GestorArchivo:
    def __init__(self, nombre_archivo):
        self.nombre = nombre_archivo
        self.lock = threading.Lock()

    def escribir_linea(self, contenido):
        with self.lock:
            with open(self.nombre, 'a') as f:
                f.write(contenido + '\n')
            time.sleep(0.1)  # Simular operación

    def leer_contenido(self):
        with self.lock:
```



```
            with open(self.nombre, 'r') as f:
                return f.readlines()

# Uso con múltiples hilos
gestor = GestorArchivo('salida.txt')

def escritor(id_escritor):
    for i in range(5):
        gestor.escribir_linea(f"Escritor {id_escritor}: Línea {i + 1}")

hilos = []
for i in range(3):
    hilo = threading.Thread(target=escritor, args=(i + 1,))
    hilos.append(hilo)
    hilo.start()

for hilo in hilos:
    hilo.join()

print("Contenido del archivo:")
print("".join(gestor.leer_contenido()))
```

3.5.3 Producer-Consumer Pattern

```
import threading
import queue
import time
import random

# Cola thread-safe para comunicación
cola = queue.Queue(maxsize=5)

def productor(id_productor):
    for i in range(5):
        item = f"Producto-{id_productor}-{i}"
        cola.put(item)
        print(f"Productor {id_productor} produjo: {item}")
        time.sleep(random.uniform(0.5, 1.5))

def consumidor(id_consumidor):
    for _ in range(5):
        item = cola.get()
        print(f"Consumidor {id_consumidor} consumió: {item}")
        time.sleep(random.uniform(0.5, 1.5))
        cola.task_done()

# Crear hilos productores y consumidores
productores = [threading.Thread(target=productor, args=(i+1,)) for i in
range(2)]
consumidores = [threading.Thread(target=consumidor, args=(i+1,)) for i in
range(2)]

for hilo in productores + consumidores:
```



```
    hilo.start()

for hilo in productores + consumidores:
    hilo.join()

print("Producción y consumo completado")
```

3.5.4 Event para coordinación

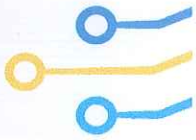
```
import threading
import time

evento = threading.Event()

def esperador(nombre):
    print(f"{nombre} esperando evento...")
    evento.wait()  # Se detiene hasta que event.set() sea llamado
    print(f"{nombre} evento recibido!")

def disparador():
    time.sleep(2)
    print("Disparando evento...")
    evento.set()

hilos = [
    threading.Thread(target=esperador, args=("Esperador 1",)),
    threading.Thread(target=esperador, args=("Esperador 2",)),
    threading.Thread(target=disparador)
]

for hilo in hilos:
    hilo.start()

for hilo in hilos:
    hilo.join()
```

3.6 EJERCICIOS RESUELTOS

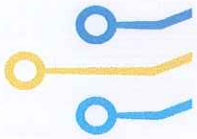
Ejercicio Resuelto 1: Gestor de Archivos CSV

Problema: Crear un programa que gestione un archivo CSV de productos con operaciones CRUD.

```
import csv
import threading

class GestorProductos:
    def __init__(self, nombre_archivo):
        self.nombre = nombre_archivo
        self.lock = threading.Lock()
        self._inicializar_archivo()

    def _inicializar_archivo(self):
        """Crear archivo con encabezados si no existe"""
        try:
```



```
            with open(self.nombre, 'r'):
                pass
        except FileNotFoundError:
            with open(self.nombre, 'w', newline='') as f:
                writer = csv.writer(f)
                writer.writerow(['ID', 'Nombre', 'Precio', 'Cantidad'])

    def crear_producto(self, id_prod, nombre, precio, cantidad):
        """CREATE: Agregar nuevo producto"""
        with self.lock:
            with open(self.nombre, 'a', newline='') as f:
                writer = csv.writer(f)
                writer.writerow([id_prod, nombre, precio, cantidad])
        print(f"Producto '{nombre}' creado exitosamente")

    def leer_productos(self):
        """READ: Obtener todos los productos"""
        with self.lock:
            with open(self.nombre, 'r') as f:
                reader = csv.DictReader(f)
                return list(reader)

    def actualizar_producto(self, id_prod, **kwargs):
        """UPDATE: Modificar un producto existente"""
        with self.lock:
            productos = self.leer_productos()

            for producto in productos:
                if producto['ID'] == str(id_prod):
                    producto.update(kwargs)
                    break

            with open(self.nombre, 'w', newline='') as f:
            writer = csv.DictWriter(f, fieldnames=['ID', 'Nombre', 'Precio',
'Cantidad'])
                writer.writeheader()
                writer.writerows(productos)
        print(f"Producto {id_prod} actualizado")

    def eliminar_producto(self, id_prod):
        """DELETE: Remover un producto"""
        with self.lock:
            productos = self.leer_productos()
            productos = [p for p in productos if p['ID'] != str(id_prod)]

            with open(self.nombre, 'w', newline='') as f:
            writer = csv.DictWriter(f, fieldnames=['ID', 'Nombre', 'Precio',
'Cantidad'])
                writer.writeheader()
                writer.writerows(productos)
        print(f"Producto {id_prod} eliminado")
```

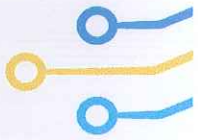


```
# Uso
gestor = GestorProductos('productos.csv')

# Crear productos
gestor.crear_producto(1, 'Laptop', 800.00, 10)
gestor.crear_producto(2, 'Mouse', 25.00, 50)
gestor.crear_producto(3, 'Teclado', 75.00, 30)

# Leer productos
print("\n--- Productos ---")
for prod in gestor.leer_productos():
    print(prod)

# Actualizar
gestor.actualizar_producto(1, Precio='750.00', Cantidad='12')

# Eliminar
# gestor.eliminar_producto(2)
```

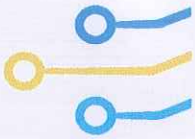
Ejercicio Resuelto 2: Descargador Multihilo

Problema: Simular descargas simultáneas usando múltiples hilos.

```
import threading
import time
import random
import json
from datetime import datetime

class DescargadorMultihilo:
    def __init__(self, archivos):
        self.archivos = archivos
        self.resultados = []
        self.lock = threading.Lock()

    def simular_descarga(self, id_archivo, nombre, tamaño_mb):
        """Simula la descarga de un archivo"""
        inicio = time.time()

        # Simular descarga con progreso
        velocidad = random.uniform(1, 5)  # MB/s
        tiempo_descarga = tamaño_mb / velocidad

        print(f"[Hilo {id_archivo}] Iniciando descarga: {nombre}
({tamaño_mb}MB)")
        time.sleep(tiempo_descarga)

        duracion = time.time() - inicio

        # Guardar resultado de forma thread-safe
        with self.lock:
            resultado = {
```



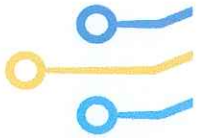
```
                'archivo': nombre,
                'tamaño_mb': tamaño_mb,
                'duracion_segundos': round(duracion, 2),
                'velocidad_mb_s': round(tamaño_mb / duracion, 2),
                'timestamp': datetime.now().strftime('%Y-%m-%d %H:%M:%S')
            }
            self.resultados.append(resultado)

        print(f"[Hilo {id_archivo}] Descarga completada: {nombre}")

    def descargar_simultaneamente(self):
        """Ejecutar descargas en paralelo"""
        hilos = []

        for idx, (nombre, tamaño) in enumerate(self.archivos, 1):
            hilo = threading.Thread(
                target=self.simular_descarga,
                args=(idx, nombre, tamaño)
            )
            hilos.append(hilo)
            hilo.start()

        # Esperar a que terminen todas
        for hilo in hilos:
            hilo.join()

    def guardar_reporte(self, nombre_reporte):
        """Guardar resultados en archivo JSON"""
        with open(nombre_reporte, 'w', encoding='utf-8') as f:
            json.dump(self.resultados, f, ensure_ascii=False, indent=2)

        total_tamaño = sum(r['tamaño_mb'] for r in self.resultados)
        total_tiempo = sum(r['duracion_segundos'] for r in self.resultados)

        print(f"\n--- Reporte de Descargas ---")
        print(f"Archivos descargados: {len(self.resultados)}")
        print(f"Tamaño total: {total_tamaño}MB")
        print(f"Tiempo total (secuencial equivalente): {total_tiempo}s")
        print(f"Reporte guardado en: {nombre_reporte}")

# Uso
archivos = [
    ('documento.pdf', 50),
    ('imagen.zip', 120),
    ('video.mp4', 250),
    ('codigo.tar.gz', 30)
]

descargador = DescargadorMultihilo(archivos)
print("Iniciando descargas simultáneas...\n")
descargador.descargar_simultaneamente()
descargador.guardar_reporte('descargas_reporte.json')
```



Ejercicio Resuelto 3: Procesador de Datos con Hilos

Problema: Procesar un archivo de datos de forma paralela.

```
import threading
import csv
import json
import time

class ProcesadorDatos:
    def __init__(self, archivo_entrada):
        self.archivo = archivo_entrada
        self.datos = []
        self.resultados = []
        self.lock = threading.Lock()

    def cargar_datos(self):
        """Cargar datos del archivo CSV"""
        with open(self.archivo, 'r', encoding='utf-8') as f:
            reader = csv.DictReader(f)
            self.datos = list(reader)
        print(f"Datos cargados: {len(self.datos)} registros")

    def procesar_registro(self, registro, id_hilo):
        """Procesar un registro individual"""
        try:
            # Simular procesamiento
            nombre = registro.get('nombre', 'Desconocido')
            edad = int(registro.get('edad', 0))
            salario = float(registro.get('salario', 0))

            # Cálculos
            categoria_edad = 'Joven' if edad < 30 else 'Adulto' if edad < 60
else 'Mayor'
            impuesto = salario * 0.20
            salario_neto = salario - impuesto

            resultado = {
                'nombre': nombre,
                'edad': edad,
                'categoria': categoria_edad,
                'salario_bruto': salario,
                'impuesto': round(impuesto, 2),
                'salario_neto': round(salario_neto, 2)
            }

            with self.lock:
                self.resultados.append(resultado)

            print(f"[Hilo {id_hilo}] Procesado: {nombre}")
            time.sleep(0.1)  # Simular trabajo

        except (ValueError, KeyError) as e:
```



```
            print(f"[Hilo {id_hilo}] Error procesando registro: {e}")

    def procesar_paralelo(self, num_hilos=4):
        """Procesar datos usando múltiples hilos"""
        tamaño_chunk = len(self.datos) // num_hilos
        hilos = []

        for i in range(num_hilos):
            inicio = i * tamaño_chunk
         fin = inicio + tamaño_chunk if i < num_hilos - 1 else len(self.datos)
            chunk = self.datos[inicio:fin]

            hilo = threading.Thread(
                target=self._procesar_chunk,
                args=(chunk, i + 1)
            )
            hilos.append(hilo)
            hilo.start()

        for hilo in hilos:
            hilo.join()

        print(f"Procesamiento completado: {len(self.resultados)} registros")

    def _procesar_chunk(self, chunk, id_hilo):
        """Procesar un chunk de datos"""
        for registro in chunk:
            self.procesar_registro(registro, id_hilo)

    def guardar_resultados(self, archivo_salida):
        """Guardar resultados en JSON"""
        with open(archivo_salida, 'w', encoding='utf-8') as f:
            json.dump(self.resultados, f, ensure_ascii=False, indent=2)
        print(f"Resultados guardados en: {archivo_salida}")

# Crear archivo de prueba
def crear_archivo_prueba():
    datos = [
        ['nombre', 'edad', 'salario'],
        ['Juan Pérez', '28', '2500'],
        ['María González', '35', '3000'],
        ['Carlos López', '45', '3500'],
        ['Ana Martínez', '22', '2000'],
        ['Pedro Sánchez', '55', '4000']
    ]

    with open('empleados.csv', 'w', newline='', encoding='utf-8') as f:
        writer = csv.writer(f)
        writer.writerows(datos)

# Uso
crear_archivo_prueba()
```

```
procesador = ProcesadorDatos('empleados.csv')
procesador.cargar_datos()
procesador.procesar_paralelo(num_hilos=2)
procesador.guardar_resultados('empleados_procesados.json')
```

3.7 EJERCICIOS PROPUESTOS

Ejercicio Propuesto 1: Sistema de Gestión de Biblioteca

Crea un sistema de gestión de biblioteca que:

- Almacene libros en un archivo JSON con campos: ID, título, autor, año, disponibilidad
- Implemente operaciones CRUD completas
- Use múltiples hilos para procesar solicitudes simultáneas de préstamo/devolución
- Sincronice el acceso al archivo con Lock
- Genere un reporte de libros disponibles

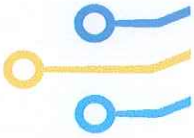
Requisitos:

- Crear al menos 10 libros iniciales
- Simular 5 hilos realizando préstamos y devoluciones simultáneamente
- Garantizar integridad de datos con sincronización
- Guardar log de operaciones en archivo de texto

Sugerencias de implementación:

python

```
class Biblioteca:
    def __init__(self, archivo_json):
        self.archivo = archivo_json
        self.lock = threading.Lock()

    def crear_libro(self, id_lib, titulo, autor, año):
        pass

    def prestar_libro(self, id_lib, usuario):
        pass

    def devolver_libro(self, id_lib):
        pass

    def listar_disponibles(self):
        pass

    def registrar_operacion(self, mensaje):
        pass
```



Ejercicio Propuesto 2: Procesador de Logs Multihilo

Nivel: Intermedio-Avanzado
Crea un procesador de logs que:

- Lea un archivo de logs con millones de líneas
- Divida el procesamiento entre múltiples hilos
- Extraiga información: errores, advertencias, información
- Agrupe por hora del día
- Genere estadísticas en JSON

Requisitos:

- Procesar logs de forma paralela (mínimo 4 hilos)
- Contar ocurrencias de cada tipo de evento
- Identificar las horas con más errores
- Calcular tiempo promedio de procesamiento
- Mostrar comparación: procesamiento secuencial vs paralelo

Formato esperado de logs:
2024-10-15 08:15:23 ERROR: Conexión fallida a base de datos
2024-10-15 08:16:45 INFO: Usuario Juan inició sesión
2024-10-15 08:17:12 WARNING: Memoria disponible baja

Ejercicio Propuesto 3: Descargador de Archivos Avanzado

Nivel: Intermedio
Crea un descargador de múltiples archivos que:

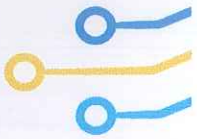
- Descargue (simule) varios archivos en paralelo
- Implemente barra de progreso para cada descarga
- Use Queue para gestionar descargas pendientes
- Permita pausar/reanudar descargas
- Registre estadísticas de cada descarga

Requisitos:

- Mínimo 5 descargas simultáneas
- Mostrar progreso en tiempo real
- Manejar errores de descarga
- Guardar resumen en archivo
- Usar Event para pausar/reanudar

Estructura sugerida:
python

```
class DescargaAvanzada:
    def __init__(self):
        self.cola_descargas = queue.Queue()
```



```
        self.pausado = threading.Event()
        self.estadisticas = []

    def agregar_descarga(self, url, destino, tamaño):
        pass

    def trabajador_descarga(self):
        pass

    def pausar_descargas(self):
        pass

    def reanudar_descargas(self):
        pass
```

Ejercicio Propuesto 4: Servidor de Chat Multihilo

Nivel: Avanzado
Crea un servidor de chat simple que:

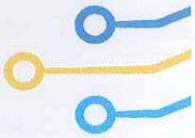
- Gestione múltiples clientes (simulados) con hilos
- Almacene mensajes en archivo
- Sincronice acceso a recursos compartidos
- Implemente comandos: /usuarios, /historial, /salir
- Genere estadísticas de uso

Requisitos:

- Simular al menos 3 clientes simultáneos
- Guardar todos los mensajes en archivo
- Mostrar lista de usuarios conectados
- Implementar notificaciones de conexión/desconexión
- Usar Lock para acceso a archivos compartidos

Estructura sugerida:
python

```
class ServidorChat:
    def __init__(self):
        self.usuarios = {}
        self.mensajes = []
        self.lock = threading.Lock()

    def conectar_usuario(self, nombre):
        pass

    def enviar_mensaje(self, usuario, contenido):
```



```
        pass

    def obtener_historial(self, limite=50):
        pass

    def desconectar_usuario(self, nombre):
        pass
```

Ejercicio Propuesto 5: Generador de Reportes Paralelo

Nivel: Intermedio

Crea un sistema que:

- Procese múltiples archivos de datos (CSV) simultáneamente
- Genere reportes individuales para cada archivo
- Agregue resultados finales en un reporte consolidado
- Use hilos para paralelizar generación de gráficos (simulada)
- Mida mejora de rendimiento

Requisitos:

- Procesar mínimo 5 archivos CSV en paralelo
- Generar reporte JSON por archivo
- Crear reporte consolidado
- Mostrar estadísticas de tiempo (secuencial vs paralelo)
- Usar sincronización para acceso a reporte consolidado

Campos a procesar:

- Cantidad de registros
- Valores mínimo, máximo, promedio
- Registros con errores
- Distribución de datos

Ejercicio Propuesto 6: Monitor de Archivos

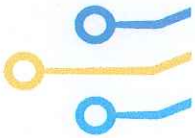
Nivel: Intermedio-Avanzado

Crea un monitor que:

- Observe cambios en un directorio
- Use hilos para monitorear múltiples directorios
- Registre cada cambio (creación, modificación, eliminación)
- Genere alertas para archivos específicos
- Resuma cambios en reporte diario

Requisitos:

- Monitorear mínimo 3 directorios
- Detectar archivos nuevos, modificados, eliminados
- Generar log de cambios



- Crear alertas para cambios importantes
- Usar sincronización para acceso a log compartido

Información a registrar:

json

```
{
 "timestamp": "2024-10-15 14:30:45",
 "directorio": "/datos",
 "tipo": "Creación",
 "archivo": "reporte.xlsx",
 "tamaño": 2048,
 "alertas": []
}
```

Ejercicio Propuesto 7: Herramienta de Búsqueda de Texto

Nivel: Intermedio

Crea una herramienta que:

- Busque un término en múltiples archivos simultáneamente
- Use un hilo por archivo
- Guarde resultados en JSON
- Genere estadísticas de búsqueda
- Implemente búsqueda por patrones (regex)

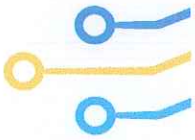
Requisitos:

- Procesar mínimo 10 archivos en paralelo
- Encontrar líneas coincidentes con contexto
- Contar ocurrencias totales
- Medir tiempo de búsqueda
- Guardar resultados con línea y posición

Estructura de resultado:

json

```
{
 "archivo": "documento.txt",
 "coincidencias": 15,
 "lineas": [
  {
   "numero": 42,
   "contenido": "...",
   "posicion_inicio": 10
  }
 ]
}
```



Ejercicio Propuesto 8: Sistema de Copias de Seguridad

Nivel: Avanzado

Crea un sistema de backup que:

- Copie múltiples directorios en paralelo
- Implemente compresión (simulada)
- Verifique integridad con checksums
- Use Queue para gestionar backups pendientes
- Genere reporte de ejecución

Requisitos:

- Backup paralelo de mínimo 4 directorios
- Crear estructura de carpetas en destino
- Calcular checksum (MD5 o similar) para verificación
- Guardar metadatos de backup
- Implementar sistema de rotación (últimas 3 copias)

Información de backup:

json

```
{
  "timestamp": "2024-10-15 23:00:00",
  "origen": "/datos",
  "destino": "/backup/2024-10-15",
  "archivos_copiados": 150,
  "tamaño_total": 5242880,
  "duracion_segundos": 45,
  "verificacion": "OK"
}
```

Notas Importantes para la Resolución

1. **Sincronización**: Siempre use `Lock` al acceder a recursos compartidos
2. **Context Managers**: Use `with` para archivos y locks
3. **Manejo de Errores**: Implemente try-except apropiadamente
4. **Nombrado de Hilos**: Asigne nombres descriptivos a los hilos para debugging
5. **Documentación**: Incluya docstrings en todas las funciones
6. **Testing**: Pruebe con datos pequeños primero, luego escale
7. **Performance**: Mida tiempos para demostrar mejora paralela
8. **Logs**: Registre todas las operaciones importantes

Conclusión

Esta unidad ha cubierto:

- Conceptos fundamentales de manejo de ficheros
- Operaciones CRUD en archivos de texto, CSV y JSON
- Conceptos de hilos y multihilo
- Sincronización y coordinación entre hilos
- Patrones avanzados como Producer-Consumer
- Aplicaciones prácticas del multihilo

El dominio de estos conceptos es esencial para crear aplicaciones robustas, escalables y eficientes que puedan procesar múltiples tareas simultáneamente.

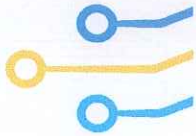
UNIDAD 4: INTERFACE GRÁFICA.

4.1 Interface gráfica
4.2 Patrón de diseño Modelo, Vistas y Controlador
4.3 Trabajar con Ficheros
4.4 Conexión a la base de datos y aplicación del CRUD
4.5 Ejercicios propuestos

Resultado de Aprendizaje

- Utiliza herramientas y tecnologías de programación para llevar a cabo tareas específicas en el campo de desarrollo de software.

DIAGRAMA DE APRENDIZAJE



SINTESIS.

Esta unidad culminante integra todos los conceptos previos en aplicaciones profesionales completas. Las **interfaces gráficas** (GUI) reemplazan la consola con ventanas, botones, menús y componentes visuales que hacen el software accesible e intuitivo. El estudiante aprende a diseñar layouts, gestionar eventos de usuario y aplicar principios de usabilidad.

El **patrón MVC** (Modelo-Vista-Controlador) estructura el código profesionalmente: el **Modelo** contiene datos y lógica de negocio, la **Vista** presenta información visualmente, y el **Controlador** coordina entre ambos. Esta separación hace el código mantenible, testeable y permite que múltiples desarrolladores trabajen simultáneamente en diferentes componentes.

La integración con **ficheros** permite importar/exportar datos desde la GUI, con diálogos de selección de archivos y feedback visual durante operaciones.

La **conexión a bases de datos** eleva las aplicaciones a sistemas empresariales. El estudiante implementa operaciones **CRUD completas** desde la interfaz gráfica: formularios para crear registros, tablas para visualizar datos, interfaces para actualizaciones y confirmaciones para eliminaciones. Aprende a usar consultas preparadas (prevenir inyección SQL), manejar transacciones (garantizar consistencia) y aplicar patrones DAO (encapsular acceso a datos).

Los ejercicios finales requieren proyectos integradores que demuestran dominio completo: GUI + MVC + Persistencia (archivos y BD) + Manejo de excepciones + todos los principios POO.

En esencia: Esta unidad transforma al estudiante de programador de ejercicios académicos en desarrollador capaz de crear aplicaciones completas, profesionales y listas para usuarios finales, cerrando el ciclo de aprendizaje con competencias aplicables inmediatamente en el mundo laboral.

UNIDAD 4: INTERFACE GRÁFICA.

Introducción

En esta unidad estudiaremos cómo crear aplicaciones de escritorio con interfaces gráficas usando Python. Pasaremos de aplicaciones de consola a aplicaciones profesionales con ventanas, botones, campos de texto y otros componentes visuales. Integraremos conceptos de diseño arquitectónico (patrón MVC), manejo de ficheros y conexión a bases de datos.

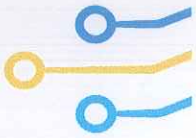
4.1 INTERFACE GRÁFICA

¿Qué es una Interface Gráfica (GUI)?

Una Interface Gráfica de Usuario (GUI - Graphical User Interface) es el medio visual a través del cual un usuario interactúa con una aplicación. En lugar de escribir comandos en consola, el usuario interactúa mediante ventanas, botones, campos de texto, menús y otros elementos visuales.

Ventajas de las interfaces gráficas

- **Usabilidad**: Más intuitiva para usuarios no técnicos
- **Profesionalismo**: Las aplicaciones lucen más completas y actualizadas
- **Interactividad**: Permite respuestas inmediatas a acciones del usuario
- **Accesibilidad**: Mejor experiencia para diferentes tipos de usuarios



Bibliotecas principales en Python

Tkinter es la biblioteca estándar de Python para crear interfaces gráficas. Es incluida por defecto en la mayoría de instalaciones de Python.

Otras opciones populares incluyen PyQt, PySimpleGUI, y Kivy, pero nos enfocaremos en Tkinter por su simpleza y accesibilidad.

Conceptos básicos de Tkinter

1. Ventana raíz (Root Window)

```
import tkinter as tk

# Crear la ventana principal
ventana = tk.Tk()
ventana.title("Mi Primera Aplicación")
ventana.geometry("400x300")  # ancho x alto

# Ejecutar la aplicación
ventana.mainloop()
```

2. Widgets (Componentes)

Los widgets son los elementos visuales de la interfaz:

- `Label`: Etiquetas de texto
- `Button`: Botones
- `Entry`: Campos de entrada de texto
- `Text`: Áreas de texto multilínea
- `Frame`: Contenedores para organizar componentes
- `Listbox`: Listas de selección
- `Combobox`: Listas desplegables
- `Messagebox`: Cuadros de diálogo

3. Gestores de diseño (Layout Managers)

- **pack()**: Apila los widgets en la ventana
- **grid()**: Organiza los widgets en una cuadrícula
- **place()**: Posiciona widgets en coordenadas específicas

Ejemplo 1: Aplicación básica con Tkinter

```
import tkinter as tk
from tkinter import messagebox

# Crear ventana
ventana = tk.Tk()
ventana.title("Aplicación Saludador")
ventana.geometry("400x200")

# Crear etiqueta
etiqueta = tk.Label(ventana, text="¿Cuál es tu nombre?", font=("Arial", 12))
etiqueta.pack(pady=10)
```

```
# Crear campo de entrada
entrada = tk.Entry(ventana, width=30, font=("Arial", 11))
entrada.pack(pady=5)

# Función para saludar
def saludar():
    nombre = entrada.get()
    if nombre:
        messagebox.showinfo("Saludo", f"¡Hola {nombre}! ¡Bienvenido!")
    else:
        messagebox.showwarning("Error", "Por favor ingresa un nombre")

# Crear botón
boton = tk.Button(ventana, text="Saludar", command=saludar, bg="blue",
fg="white")
boton.pack(pady=10)

ventana.mainloop()
```

4.2 PATRÓN DE DISEÑO MODELO-VISTA-CONTROLADOR (MVC)

¿Qué es el patrón MVC?

El patrón MVC divide una aplicación en tres componentes independientes que se comunican entre sí:

1. **Modelo (Model)**: Contiene la lógica de negocio y los datos. Es independiente de la interfaz gráfica.
2. **Vista (View)**: Presentación visual de los datos. Es responsable de mostrar información al usuario.
3. **Controlador (Controller)**: Maneja la interacción del usuario y comunica la vista con el modelo.

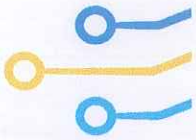
Ventajas del patrón MVC

- **Separación de responsabilidades**: Cada componente tiene un rol específico
- **Mantenibilidad**: Cambios en la interfaz no afectan la lógica de negocio
- **Reutilización**: El modelo puede usarse con diferentes interfaces
- **Testabilidad**: Cada componente puede probarse independientemente
- **Escalabilidad**: Facilita el crecimiento de la aplicación

Flujo del patrón MVC

```
Usuario → Vista → Controlador → Modelo
          ↑                        ↓
          ←————————————————————←
```

El usuario interactúa con la Vista, que envía eventos al Controlador. El Controlador procesa estos eventos y actualiza el Modelo. El Modelo notifica a la Vista sobre cambios, que se actualiza para mostrar la nueva información.



Ejemplo 2: Aplicación MVC - Gestor de tareas

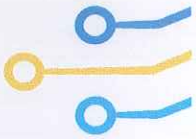
Crearemos una aplicación para gestionar tareas (crear, eliminar, listar) usando el patrón MVC.

Archivo: modelo.py

```
# MODELO: Contiene la lógica de negocio
class Tarea:
    def __init__(self, id, titulo, descripcion, completada=False):
        self.id = id
        self.titulo = titulo
        self.descripcion = descripcion
        self.completada = completada

class ModeloTareas:
    def __init__(self):
        self.tareas = []
        self.contador_id = 1
        self.observadores = []

    def agregar_observador(self, observador):
        """Observer pattern para notificar cambios"""
        self.observadores.append(observador)

    def notificar_observadores(self):
        """Notifica a los observadores sobre cambios"""
        for observador in self.observadores:
            observador.actualizar()

    def crear_tarea(self, titulo, descripcion):
        nueva_tarea = Tarea(self.contador_id, titulo, descripcion)
        self.tareas.append(nueva_tarea)
        self.contador_id += 1
        self.notificar_observadores()
        return nueva_tarea

    def obtener_tareas(self):
        return self.tareas

    def eliminar_tarea(self, id_tarea):
        self.tareas = [t for t in self.tareas if t.id != id_tarea]
        self.notificar_observadores()

    def marcar_completada(self, id_tarea):
        for tarea in self.tareas:
            if tarea.id == id_tarea:
                tarea.completada = not tarea.completada
                self.notificar_observadores()
                break
```

Archivo: controlador.py

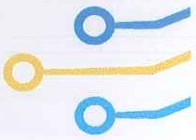
```
# CONTROLADOR: Maneja la comunicación entre Vista y Modelo
class ControladorTareas:
```



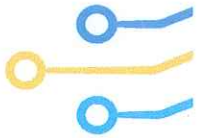
```
    def __init__(self, modelo):
        self.modelo = modelo

    def agregar_tarea(self, titulo, descripcion):
        if not titulo or not descripcion:
            return False
        self.modelo.crear_tarea(titulo, descripcion)
        return True

    def obtener_todas_tareas(self):
        return self.modelo.obtener_tareas()

    def eliminar_tarea(self, id_tarea):
        self.modelo.eliminar_tarea(id_tarea)

    def completar_tarea(self, id_tarea):
        self.modelo.marcar_completada(id_tarea)

    def obtener_tarea_por_id(self, id_tarea):
        for tarea in self.modelo.obtener_tareas():
            if tarea.id == id_tarea:
                return tarea
        return None
```

Archivo: vista.py

```
import tkinter as tk
from tkinter import messagebox, ttk
from controlador import ControladorTareas
from modelo import ModeloTareas

# VISTA: Interfaz gráfica
class VistaTareas:
    def __init__(self, ventana_raiz):
        self.ventana = ventana_raiz
        self.ventana.title("Gestor de Tareas MVC")
        self.ventana.geometry("600x500")

        # Inicializar modelo y controlador
        self.modelo = ModeloTareas()
        self.controlador = ControladorTareas(self.modelo)

        # Registrar esta vista como observadora del modelo
        self.modelo.agregar_observador(self)

        self.crear_interfaz()

    def crear_interfaz(self):
        """Crea todos los componentes visuales"""

        # Frame superior para entrada
        frame_entrada = tk.Frame(self.ventana, bg="lightblue", pady=10)
        frame_entrada.pack(fill=tk.X)
```



```
        # Campos de entrada
      tk.Label(frame_entrada, text="Título:", bg="lightblue", font=("Arial",
10)).pack(side=tk.LEFT, padx=5)
      self.entrada_titulo = tk.Entry(frame_entrada, width=20, font=("Arial",
10))
        self.entrada_titulo.pack(side=tk.LEFT, padx=5)

        tk.Label(frame_entrada, text="Descripción:", bg="lightblue",
font=("Arial", 10)).pack(side=tk.LEFT, padx=5)
        self.entrada_descripcion = tk.Entry(frame_entrada, width=20,
font=("Arial", 10))
        self.entrada_descripcion.pack(side=tk.LEFT, padx=5)

        # Botón agregar
        boton_agregar = tk.Button(frame_entrada, text="Agregar",
command=self.agregar_tarea, bg="green", fg="white")
        boton_agregar.pack(side=tk.LEFT, padx=5)

        # Frame para la lista de tareas
        frame_lista = tk.Frame(self.ventana, pady=10)
        frame_lista.pack(fill=tk.BOTH, expand=True, padx=10)

        tk.Label(frame_lista, text="Tareas:", font=("Arial", 12,
"bold")).pack(anchor=tk.W)

        # Listbox para mostrar tareas
        self.lista_tareas = tk.Listbox(frame_lista, font=("Arial", 10),
height=15)
        self.lista_tareas.pack(fill=tk.BOTH, expand=True, pady=5)
        self.lista_tareas.bind('<<ListboxSelect>>', self.seleccionar_tarea)

        # Frame para botones de acción
        frame_botones = tk.Frame(self.ventana, pady=10)
        frame_botones.pack(fill=tk.X)

        boton_completar = tk.Button(frame_botones, text="Marcar Completada",
command=self.completar_tarea, bg="orange")
        boton_completar.pack(side=tk.LEFT, padx=5)

        boton_eliminar = tk.Button(frame_botones, text="Eliminar",
command=self.eliminar_tarea, bg="red", fg="white")
        boton_eliminar.pack(side=tk.LEFT, padx=5)

        self.tarea_seleccionada = None

    def agregar_tarea(self):
        """Maneja la adición de una nueva tarea"""
        titulo = self.entrada_titulo.get()
        descripcion = self.entrada_descripcion.get()

        if self.controlador.agregar_tarea(titulo, descripcion):
```



```
            self.entrada_titulo.delete(0, tk.END)
            self.entrada_descripcion.delete(0, tk.END)
        else:
         messagebox.showerror("Error", "Por favor completa todos los campos")

    def seleccionar_tarea(self, evento):
        """Guarda la tarea seleccionada"""
        seleccion = self.lista_tareas.curselection()
        if seleccion:
            tareas = self.controlador.obtener_todas_tareas()
            self.tarea_seleccionada = tareas[seleccion[0]]

    def completar_tarea(self):
        """Marca una tarea como completada"""
        if self.tarea_seleccionada:
            self.controlador.completar_tarea(self.tarea_seleccionada.id)
        else:
            messagebox.showwarning("Advertencia", "Selecciona una tarea")

    def eliminar_tarea(self):
        """Elimina la tarea seleccionada"""
        if self.tarea_seleccionada:
            self.controlador.eliminar_tarea(self.tarea_seleccionada.id)
            self.tarea_seleccionada = None
        else:
            messagebox.showwarning("Advertencia", "Selecciona una tarea")

    def actualizar(self):
        """Actualiza la vista cuando el modelo cambia (Observer Pattern)"""
        self.lista_tareas.delete(0, tk.END)
        tareas = self.controlador.obtener_todas_tareas()
        for tarea in tareas:
            estado = "✓ " if tarea.completada else "o "
            self.lista_tareas.insert(tk.END,    f"{estado}{tarea.titulo}    -
{tarea.descripcion}")

# Archivo principal: main.py
if __name__ == "__main__":
    ventana = tk.Tk()
    app = VistaTareas(ventana)
    ventana.mainloop()
```

4.3 TRABAJAR CON FICHEROS

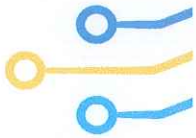
¿Por qué trabajar con ficheros?

Las aplicaciones necesitan persistencia de datos. Trabajar con ficheros nos permite guardar información que se mantiene incluso después de cerrar la aplicación.

Operaciones básicas con ficheros en Python

1. Lectura de ficheros

```
# Modo lectura
with open('archivo.txt', 'r', encoding='utf-8') as archivo:
```



```
    contenido = archivo.read()  # Leer todo el contenido
    # O leer línea por línea
    # for linea in archivo:
    #     print(linea)
```

2. Escritura de ficheros

```
# Modo escritura (sobrescribe si existe)
with open('archivo.txt', 'w', encoding='utf-8') as archivo:
    archivo.write("Contenido nuevo\n")
    archivo.write("Otra línea\n")

# Modo adición (añade al final)
with open('archivo.txt', 'a', encoding='utf-8') as archivo:
    archivo.write("Contenido adicional\n")
```

3. Trabajar con JSON

JSON es un formato muy popular para almacenar datos estructurados.

```
import json

# Guardar datos en JSON
datos = {
    "nombre": "Juan",
    "edad": 30,
    "tareas": ["Tarea 1", "Tarea 2"]
}

with open('datos.json', 'w', encoding='utf-8') as archivo:
    json.dump(datos, archivo, indent=4, ensure_ascii=False)

# Cargar datos desde JSON
with open('datos.json', 'r', encoding='utf-8') as archivo:
    datos_cargados = json.load(archivo)
    print(datos_cargados)
```

4. Trabajar con CSV

```
import csv

# Escribir CSV
tareas = [
    {"id": 1, "titulo": "Tarea 1", "completada": False},
    {"id": 2, "titulo": "Tarea 2", "completada": True}
]

with open('tareas.csv', 'w', newline='', encoding='utf-8') as archivo:
   writer = csv.DictWriter(archivo, fieldnames=["id", "titulo", "completada"])
    writer.writeheader()
    writer.writerows(tareas)

# Leer CSV
with open('tareas.csv', 'r', encoding='utf-8') as archivo:
    reader = csv.DictReader(archivo)
```

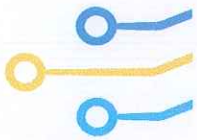
```
    for fila in reader:
        print(fila)
```

Ejemplo 3: Aplicación MVC mejorada con persistencia en ficheros

Modificaremos el modelo para guardar tareas en JSON.

Archivo: modelo_persistente.py

```
import json
import os
from datetime import datetime

class Tarea:
    def __init__(self, id, titulo, descripcion, completada=False,
fecha_creacion=None):
        self.id = id
        self.titulo = titulo
        self.descripcion = descripcion
        self.completada = completada
        self.fecha_creacion = fecha_creacion or
datetime.now().strftime("%Y-%m-%d %H:%M:%S")

class ModeloTareasConPersistencia:
    def __init__(self, nombre_archivo="tareas.json"):
        self.nombre_archivo = nombre_archivo
        self.tareas = []
        self.contador_id = 1
        self.observadores = []
        self.cargar_tareas()

    def cargar_tareas(self):
        """Carga tareas desde el fichero JSON"""
        if os.path.exists(self.nombre_archivo):
            try:
            with open(self.nombre_archivo, 'r', encoding='utf-8') as archivo:
                    datos = json.load(archivo)
                    self.tareas = [
                        Tarea(t['id'], t['titulo'], t['descripcion'],
                              t['completada'], t['fecha_creacion'])
                        for t in datos
                    ]
                    if self.tareas:
                        self.contador_id = max(t.id for t in self.tareas) + 1
            except:
                self.tareas = []

    def guardar_tareas(self):
        """Guarda tareas en el fichero JSON"""
        datos = [
            {
                'id': t.id,
                'titulo': t.titulo,
                'descripcion': t.descripcion,
```



```
                'completada': t.completada,
                'fecha_creacion': t.fecha_creacion
            }
            for t in self.tareas
        ]
        with open(self.nombre_archivo, 'w', encoding='utf-8') as archivo:
            json.dump(datos, archivo, indent=4, ensure_ascii=False)

    def agregar_observador(self, observador):
        self.observadores.append(observador)

    def notificar_observadores(self):
        for observador in self.observadores:
            observador.actualizar()

    def crear_tarea(self, titulo, descripcion):
        nueva_tarea = Tarea(self.contador_id, titulo, descripcion)
        self.tareas.append(nueva_tarea)
        self.contador_id += 1
        self.guardar_tareas()
        self.notificar_observadores()
        return nueva_tarea

    def obtener_tareas(self):
        return self.tareas

    def eliminar_tarea(self, id_tarea):
        self.tareas = [t for t in self.tareas if t.id != id_tarea]
        self.guardar_tareas()
        self.notificar_observadores()

    def marcar_completada(self, id_tarea):
        for tarea in self.tareas:
            if tarea.id == id_tarea:
                tarea.completada = not tarea.completada
                self.guardar_tareas()
                self.notificar_observadores()
                break
```

4.4 CONEXIÓN A BASE DE DATOS Y APLICACIÓN DE CRUD

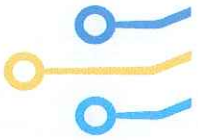
¿Qué es CRUD?

CRUD representa las cuatro operaciones básicas en persistencia de datos:

- **Create (Crear)**: Insertar nuevos registros
- **Read (Leer)**: Consultar registros existentes
- **Update (Actualizar)**: Modificar registros
- **Delete (Eliminar)**: Borrar registros

Instalación de SQLite y conexión

SQLite es un motor de base de datos ligero incluido en Python.

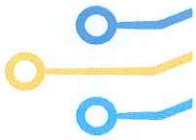


```
import sqlite3

# Conexión a la base de datos (crea el archivo si no existe)
conexion = sqlite3.connect('mibasedatos.db')
cursor = conexion.cursor()

# Crear tabla
cursor.execute('''
    CREATE TABLE IF NOT EXISTS tareas (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        titulo TEXT NOT NULL,
        descripcion TEXT NOT NULL,
        completada BOOLEAN DEFAULT 0,
        fecha_creacion TIMESTAMP DEFAULT CURRENT_TIMESTAMP
    )
''')

conexion.commit()
cursor.close()
conexion.close()
```

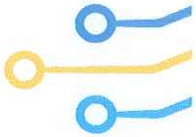
Operaciones CRUD completas

Archivo: base_datos.py

```
import sqlite3
from contextlib import contextmanager
from datetime import datetime

class BaseDatos:
    def __init__(self, nombre_bd='tareas.db'):
        self.nombre_bd = nombre_bd
        self.inicializar_bd()

    def get_conexion(self):
        """Retorna una conexión a la base de datos"""
        conexion = sqlite3.connect(self.nombre_bd)
        conexion.row_factory = sqlite3.Row  # Acceder columnas por nombre
        return conexion

    def inicializar_bd(self):
        """Crea la tabla de tareas si no existe"""
        conexion = self.get_conexion()
        cursor = conexion.cursor()

        cursor.execute('''
            CREATE TABLE IF NOT EXISTS tareas (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                titulo TEXT NOT NULL,
                descripcion TEXT NOT NULL,
                completada BOOLEAN DEFAULT 0,
                fecha_creacion TIMESTAMP DEFAULT CURRENT_TIMESTAMP
            )
        ''')
```



```
        conexion.commit()
        cursor.close()
        conexion.close()

    # CREATE
    def crear_tarea(self, titulo, descripcion):
        """Inserta una nueva tarea en la base de datos"""
        conexion = self.get_conexion()
        cursor = conexion.cursor()

        try:
            cursor.execute(
                'INSERT INTO tareas (titulo, descripcion) VALUES (?, ?)',
                (titulo, descripcion)
            )
            conexion.commit()
            id_tarea = cursor.lastrowid
            return {'id': id_tarea, 'exito': True}
        except sqlite3.Error as e:
            return {'id': None, 'exito': False, 'error': str(e)}
        finally:
            cursor.close()
            conexion.close()

    # READ
    def obtener_todas_tareas(self):
        """Obtiene todas las tareas de la base de datos"""
        conexion = self.get_conexion()
        cursor = conexion.cursor()

        try:
            cursor.execute('SELECT * FROM tareas')
            tareas = cursor.fetchall()
            return [dict(tarea) for tarea in tareas]
        finally:
            cursor.close()
            conexion.close()

    def obtener_tarea_por_id(self, id_tarea):
        """Obtiene una tarea específica por su ID"""
        conexion = self.get_conexion()
        cursor = conexion.cursor()

        try:
            cursor.execute('SELECT * FROM tareas WHERE id = ?', (id_tarea,))
            tarea = cursor.fetchone()
            return dict(tarea) if tarea else None
        finally:
            cursor.close()
            conexion.close()
```



```
    # UPDATE
    def actualizar_tarea(self, id_tarea, titulo, descripcion):
        """Actualiza los datos de una tarea"""
        conexion = self.get_conexion()
        cursor = conexion.cursor()

        try:
            cursor.execute(
                'UPDATE tareas SET titulo = ?, descripcion = ? WHERE id = ?',
                (titulo, descripcion, id_tarea)
            )
            conexion.commit()
            return {'exito': cursor.rowcount > 0}
        except sqlite3.Error as e:
            return {'exito': False, 'error': str(e)}
        finally:
            cursor.close()
            conexion.close()

    def marcar_completada(self, id_tarea):
        """Cambia el estado de completitud de una tarea"""
        conexion = self.get_conexion()
        cursor = conexion.cursor()

        try:
            cursor.execute(
                'UPDATE tareas SET completada = NOT completada WHERE id = ?',
                (id_tarea,)
            )
            conexion.commit()
            return {'exito': cursor.rowcount > 0}
        except sqlite3.Error as e:
            return {'exito': False, 'error': str(e)}
        finally:
            cursor.close()
            conexion.close()

    # DELETE
    def eliminar_tarea(self, id_tarea):
        """Elimina una tarea de la base de datos"""
        conexion = self.get_conexion()
        cursor = conexion.cursor()

        try:
            cursor.execute('DELETE FROM tareas WHERE id = ?', (id_tarea,))
            conexion.commit()
            return {'exito': cursor.rowcount > 0}
        except sqlite3.Error as e:
            return {'exito': False, 'error': str(e)}
        finally:
            cursor.close()
            conexion.close()
```

```
    def eliminar_todas_tareas(self):
        """Elimina todas las tareas"""
        conexion = self.get_conexion()
        cursor = conexion.cursor()

        try:
            cursor.execute('DELETE FROM tareas')
            conexion.commit()
            return {'exito': True}
        except sqlite3.Error as e:
            return {'exito': False, 'error': str(e)}
        finally:
            cursor.close()
            conexion.close()
```

Ejemplo 4: Aplicación MVC completa con base de datos

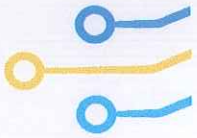
Archivo: controlador_bd.py

```
from base_datos import BaseDatos

class ControladorTareasConBD:
    def __init__(self):
        self.bd = BaseDatos()

    def agregar_tarea(self, titulo, descripcion):
        """Valida y crea una nueva tarea"""
        if not titulo or not titulo.strip():
            return {'exito': False, 'mensaje': 'El título es requerido'}
        if not descripcion or not descripcion.strip():
            return {'exito': False, 'mensaje': 'La descripción es requerida'}

        resultado = self.bd.crear_tarea(titulo.strip(), descripcion.strip())
        if resultado['exito']:
            return {'exito': True, 'mensaje': 'Tarea creada exitosamente'}
        else:
         return {'exito': False, 'mensaje': f"Error: {resultado.get('error',
'desconocido')}"}

    def obtener_tareas(self):
        """Obtiene todas las tareas"""
        return self.bd.obtener_todas_tareas()

    def eliminar_tarea(self, id_tarea):
        """Elimina una tarea"""
        resultado = self.bd.eliminar_tarea(id_tarea)
        if resultado['exito']:
            return {'exito': True, 'mensaje': 'Tarea eliminada'}
        else:
            return {'exito': False, 'mensaje': 'Error al eliminar'}

    def completar_tarea(self, id_tarea):
        """Marca una tarea como completada"""
        resultado = self.bd.marcar_completada(id_tarea)
```

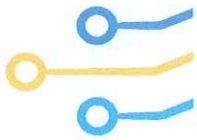
```
        if resultado['exito']:
            return {'exito': True}
        else:
            return {'exito': False, 'mensaje': 'Error al actualizar'}

    def actualizar_tarea(self, id_tarea, titulo, descripcion):
        """Actualiza los datos de una tarea"""
        resultado = self.bd.actualizar_tarea(id_tarea, titulo, descripcion)
        if resultado['exito']:
            return {'exito': True, 'mensaje': 'Tarea actualizada'}
        else:
            return {'exito': False, 'mensaje': 'Error al actualizar'}
```

Archivo: vista_completa.py

```
import tkinter as tk
from tkinter import messagebox, ttk
from controlador_bd import ControladorTareasConBD

class VistaCompleta:
    def __init__(self, ventana_raiz):
        self.ventana = ventana_raiz
        self.ventana.title("Gestor de Tareas - BD")
        self.ventana.geometry("800x600")

        self.controlador = ControladorTareasConBD()
        self.tarea_seleccionada = None
        self.crear_interfaz()
        self.actualizar_lista()

    def crear_interfaz(self):
        """Crea la interfaz gráfica completa"""

        # Frame superior
        frame_entrada = tk.Frame(self.ventana, bg="lightblue", pady=10)
        frame_entrada.pack(fill=tk.X)

        tk.Label(frame_entrada, text="Título:",
bg="lightblue").pack(side=tk.LEFT, padx=5)
        self.entrada_titulo = tk.Entry(frame_entrada, width=25)
        self.entrada_titulo.pack(side=tk.LEFT, padx=5)

        tk.Label(frame_entrada, text="Descripción:",
bg="lightblue").pack(side=tk.LEFT, padx=5)
        self.entrada_descripcion = tk.Entry(frame_entrada, width=25)
        self.entrada_descripcion.pack(side=tk.LEFT, padx=5)

        boton_agregar = tk.Button(frame_entrada, text="Agregar",
command=self.agregar_tarea, bg="green", fg="white")
        boton_agregar.pack(side=tk.LEFT, padx=5)

        # Frame para la tabla
        frame_tabla = tk.Frame(self.ventana)
        frame_tabla.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)

        tk.Label(frame_tabla, text="Tareas:", font=("Arial", 12,
"bold")).pack(anchor=tk.W)

        # Tabla con Treeview
```



```
        self.tabla = ttk.Treeview(
            frame_tabla,
            columns=("ID", "Título", "Descripción", "Estado", "Fecha"),
            height=15
        )

        self.tabla.column("#0", width=0, stretch=tk.NO)
        self.tabla.column("ID", anchor=tk.CENTER, width=40)
        self.tabla.column("Título", anchor=tk.W, width=150)
        self.tabla.column("Descripción", anchor=tk.W, width=250)
        self.tabla.column("Estado", anchor=tk.CENTER, width=80)
        self.tabla.column("Fecha", anchor=tk.CENTER, width=120)

        self.tabla.heading("#0", text="", anchor=tk.W)
        self.tabla.heading("ID", text="ID", anchor=tk.CENTER)
        self.tabla.heading("Título", text="Título", anchor=tk.W)
        self.tabla.heading("Descripción", text="Descripción", anchor=tk.W)
        self.tabla.heading("Estado", text="Estado", anchor=tk.CENTER)
        self.tabla.heading("Fecha", text="Fecha", anchor=tk.CENTER)

        self.tabla.pack(fill=tk.BOTH, expand=True, pady=5)
        self.tabla.bind('<<TreeviewSelect>>', self.seleccionar_fila)

        # Frame de botones
        frame_botones = tk.Frame(self.ventana, pady=10)
        frame_botones.pack(fill=tk.X)

        boton_completar = tk.Button(frame_botones, text="Marcar Completada",
command=self.completar_tarea, bg="orange")
        boton_completar.pack(side=tk.LEFT, padx=5)

        boton_eliminar = tk.Button(frame_botones, text="Eliminar",
command=self.eliminar_tarea, bg="red", fg="white")
        boton_eliminar.pack(side=tk.LEFT, padx=5)

        boton_actualizar = tk.Button(frame_botones, text="Actualizar",
command=self.actualizar_tarea, bg="blue", fg="white")
        boton_actualizar.pack(side=tk.LEFT, padx=5)

        boton_refrescar = tk.Button(frame_botones, text="Refrescar",
command=self.actualizar_lista, bg="purple", fg="white")
        boton_refrescar.pack(side=tk.LEFT, padx=5)

    def agregar_tarea(self):
        """Agrega una nueva tarea"""
        titulo = self.entrada_titulo.get()
        descripcion = self.entrada_descripcion.get()

        resultado = self.controlador.agregar_tarea(titulo, descripcion)
        if resultado['exito']:
            messagebox.showinfo("Éxito", resultado['mensaje'])
            self.entrada_titulo.delete(0, tk.END)
            self.entrada_descripcion.delete(0, tk.END)
            self.actualizar_lista()
        else:
            messagebox.showerror("Error", resultado['mensaje'])

    def seleccionar_fila(self, evento):
        """Maneja la selección de una fila"""
        seleccion = self.tabla.selection()
```



```
        if seleccion:
            item = seleccion[0]
            valores = self.tabla.item(item)['values']
            self.tarea_seleccionada = valores[0]  # ID

    def completar_tarea(self):
        """Marca una tarea como completada"""
        if not self.tarea_seleccionada:
            messagebox.showwarning("Advertencia", "Selecciona una tarea")
            return

        resultado = self.controlador.completar_tarea(self.tarea_seleccionada)
        if resultado['exito']:
            self.actualizar_lista()
        else:
            messagebox.showerror("Error", resultado.get('mensaje', 'Error
desconocido'))

    def eliminar_tarea(self):
        """Elimina la tarea seleccionada"""
        if not self.tarea_seleccionada:
            messagebox.showwarning("Advertencia", "Selecciona una tarea")
            return

        if messagebox.askyesno("Confirmar", "¿Deseas eliminar esta tarea?"):
            resultado =
self.controlador.eliminar_tarea(self.tarea_seleccionada)
            if resultado['exito']:
                messagebox.showinfo("Éxito", resultado['mensaje'])
                self.actualizar_lista()
                self.tarea_seleccionada = None
            else:
                messagebox.showerror("Error", resultado['mensaje'])

    def actualizar_tarea(self):
        """Actualiza la tarea seleccionada"""
        if not self.tarea_seleccionada:
            messagebox.showwarning("Advertencia", "Selecciona una tarea")
            return

        titulo = self.entrada_titulo.get()
        descripcion = self.entrada_descripcion.get()

        if not titulo or not descripcion:
            messagebox.showwarning("Advertencia", "Completa todos los
campos")
            return

        resultado =
self.controlador.actualizar_tarea(self.tarea_seleccionada, titulo,
descripcion)
        if resultado['exito']:
            messagebox.showinfo("Éxito", resultado['mensaje'])
            self.actualizar_lista()
            self.entrada_titulo.delete(0, tk.END)
            self.entrada_descripcion.delete(0, tk.END)
        else:
            messagebox.showerror("Error", resultado['mensaje'])

    def actualizar_lista(self):
```

```
        """Recarga la lista de tareas desde la base de datos"""
        # Limpiar tabla
        for item in self.tabla.get_children():
            self.tabla.delete(item)

        # Cargar tareas
        tareas = self.controlador.obtener_tareas()
        for tarea in tareas:
            estado = "✓ Completada" if tarea['completada'] else "Pendiente"
            fecha = tarea['fecha_creacion'][:10] if tarea['fecha_creacion']
else ""

            self.tabla.insert('', 'end', values=(
                tarea['id'],
                tarea['titulo'],
                tarea['descripcion'],
                estado,
                fecha
            ))

# Archivo: main.py
if __name__ == "__main__":
    ventana = tk.Tk()
    app = VistaCompleta(ventana)
    ventana.mainloop()
```

4.5 EJERCICIOS PROPUESTOS

Nivel 1: Ejercicios Básicos

Ejercicio 1.1: Calculadora Simple

Crea una aplicación con interfaz gráfica que implemente una calculadora simple con las operaciones básicas (suma, resta, multiplicación, división).

Requisitos:

- Campo de entrada para dos números
- Botones para cada operación
- Mostrar el resultado en una etiqueta
- Manejo de errores (división por cero)

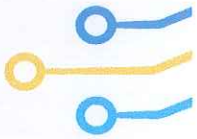
Pistas: Usa Entry para los números, Button para las operaciones, y Label para mostrar resultados.

Ejercicio 1.2: Formulario de Registro

Crea un formulario con los siguientes campos:

- Nombre (Entry)
- Email (Entry)
- Teléfono (Entry)
- Género (Radiobutton)
- País (Combobox)

Requisitos:



- Validar que todos los campos estén completos
- Mostrar un mensaje de confirmación
- Limpiar el formulario después de enviar

Ejercicio 1.3: Converter de Unidades

Crea una aplicación que convierta entre diferentes unidades (metros a kilómetros, celsius a fahrenheit, etc.)

Requisitos:

- Desplegable para seleccionar tipo de conversión
- Campo de entrada
- Botón para convertir
- Mostrar resultado

Nivel 2: Ejercicios Intermedios

Ejercicio 2.1: Gestor de Contactos (Archivos)

Crea una aplicación que:

- Agregue contactos (nombre, teléfono, email)
- Guarde los contactos en un archivo JSON
- Cargue contactos al iniciar
- Permita buscar contactos
- Permita eliminar contactos

Estructura sugerida:

```
# modelo_contactos.py
class ModeloContactos:
    def agregar_contacto(self, nombre, telefono, email)
    def obtener_contactos(self)
    def eliminar_contacto(self, nombre)
    def buscar_contacto(self, nombre)
    def guardar_en_json(self)
    def cargar_de_json(self)
```

Ejercicio 2.2: Implementar el patrón MVC en Gestor de Notas

Crea una aplicación de notas usando el patrón MVC completo:

Modelo:

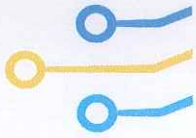
- Clase Nota con (id, titulo, contenido, fecha_creacion)
- Clase ModeloNotas con CRUD

Controlador:

- ControladorNotas para comunicar Vista y Modelo

Vista:

- Interfaz gráfica con Tkinter
- Mostrar lista de notas
- Crear, editar y eliminar notas



- Búsqueda de notas

Ejercicio 2.3: Aplicación de Tareas con Persistencia en Ficheros CSV

Modifica el gestor de tareas para guardar en CSV en lugar de JSON.

Requisitos:

- Guardar y cargar desde CSV
- Mostrar tareas en tabla
- CRUD completo
- Exportar a CSV

Nivel 3: Ejercicios Avanzados

Ejercicio 3.1: Sistema CRUD Completo con Base de Datos - Biblioteca

Crea un sistema de gestión de biblioteca con:

Tablas:

- Libros (id, titulo, autor, isbn, año, disponible)
- Prestamos (id, id_libro, fecha_prestamo, fecha_devolucion)

Funcionalidades:

- Registrar libros
- Realizar préstamos
- Registrar devoluciones
- Consultar libros disponibles
- Listar historial de préstamos
- Interfaz gráfica completa (MVC)

Estructura sugerida:

python

```
# base_datos_biblioteca.py
class BaseDatosBiblioteca:
    def crear_libro()
    def obtener_libros()
    def actualizar_disponibilidad()
    def crear_prestamo()
    def obtener_prestamos()
    def registrar_devolucion()

# controlador_biblioteca.py
class ControladorBiblioteca:
    # Lógica de validación y coordinación

# vista_biblioteca.py
class VistaBiblioteca:
    # Interfaz gráfica con pestañas para:
    # - Gestión de libros
    # - Prestamos
```

Ejercicio 3.2: Aplicación de Inventario con Reportes

Crea un sistema de gestión de inventario que incluya:

Tablas:

- Productos (id, nombre, precio, cantidad, categoria)
- Movimientos (id, id_producto, tipo, cantidad, fecha)

Funcionalidades:

- CRUD de productos
- Registrar entradas y salidas
- Alertas de stock bajo
- Generar reportes en PDF o CSV
- Gráficos de movimientos
- Búsqueda y filtrado avanzado

Ejercicio 3.3: Proyecto Integrador - Sistema de Gestión Académica

Crea una aplicación completa que incluya:

Entidades:

- Estudiantes
- Cursos
- Calificaciones
- Asistencia

Funcionalidades:

- Registrar estudiantes y cursos
- Asignar calificaciones
- Registrar asistencia
- Generar reportes (promedio, ranking)
- Búsqueda avanzada
- Exportar datos

Requisitos técnicos:

- Aplicar patrón MVC estrictamente
- Base de datos SQLite
- Interfaz profesional
- Validación robusta
- Manejo de errores
- Documentación del código

Soluciones Recomendadas

Para los ejercicios de Nivel 1 y 2:

1. Comienza con los requisitos básicos
2. Agrega validación de entrada
3. Implementa manejo de errores
4. Mejora la interfaz visual

Para los ejercicios de Nivel 3:

1. Diseña primero la estructura de la base de datos
2. Implementa el modelo con todas las operaciones
3. Crea el controlador con la lógica de validación
4. Desarrolla la vista con una interfaz clara
5. Prueba cada componente independientemente
6. Integra todo el sistema

Consejos de Implementación

Buenas prácticas:

- Usa nombres descriptivos para variables y funciones
- Implementa validación de entrada en todos los formularios
- Separa claramente el código en módulos (modelo, vista, controlador)
- Usa excepciones para manejo de errores
- Comenta tu código explicando la lógica compleja
- Prueba tu aplicación con diferentes escenarios

Estructura de carpetas recomendada:

```
mi_aplicacion/
├── main.py
├── modelo.py
├── controlador.py
├── vista.py
├── base_datos.py
├── datos/
│   └── aplicacion.db
└── recursos/
    └── estilos.txt
```

Recursos Adicionales

Documentación oficial:

- Tkinter: https://docs.python.org/3/library/tkinter.html
- SQLite: https://docs.python.org/3/library/sqlite3.html
- JSON: https://docs.python.org/3/library/json.html

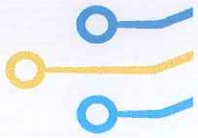
Libros y tutoriales recomendados:

- "Python GUI Programming with Tkinter" (Alan D. Moore)
- Tutoriales oficiales en python.org
- Documentación de Flask para web alternativa

Conclusión

En esta unidad has aprendido:

- ✓ Crear interfaces gráficas con Tkinter



- ✓ Aplicar el patrón MVC en aplicaciones reales
- ✓ Trabajar con ficheros (JSON, CSV)
- ✓ Conectar aplicaciones a bases de datos SQLite
- ✓ Implementar operaciones CRUD completas
- ✓ Validar entrada de usuarios
- ✓ Manejar errores de forma profesional

Estos conocimientos te permiten crear aplicaciones de escritorio robustas, mantenibles y escalables.

ELABORACIÓN, REVISIÓN Y APROBACIÓN DE PARES	
Profesor(a)	
Ing. Juan Marcial Espín Montesdeoca	
Fecha de elaboración: 31/10/2025	
Comisión de revisión de pares de guías de estudio del Instituto Superior Tecnológico Tena	
Lcda. Maria Angélica Campoverde Encalada	Mg. Alvaro Santiago Toalombo Díaz
Mg. Henry Fabian Chango Chango	Mg. Duarte Mora Martha Janina
Abg. Danilo Alexander Zamora Núñez., Mg.	
Fecha de revisión: 28/11/2025	
Coordinador de Investigación, Desarrollo Tecnológico e Innovación	
Abg. Danilo Alexander Zamora Núñez., Mg.	
Fecha de aprobación: 09/12/2025	