Tutorial Completo de Python - Básico e Intermedio

✅ MANIPULACION Y PRESERVACION DE DATOS - PROFE: AGUSTIN GATICA

🎯 NIVEL BÁSICO (Fundamentos estructurales)

1️⃣ Sintaxis y Estructura Básica

Comentarios

Los comentarios son líneas que el intérprete ignora. Sirven para documentar el código.

```
# Esto es un comentario de una línea

"""
Esto es un comentario de múltiples líneas
usando comillas triples. Se usa frecuentemente
como documentación de funciones y clases.
"""

print("Hola, Mundo!")  # Este también es un comentario
```

Indentación / Bloques

Python usa **indentación** (espacios en blanco) para definir bloques de código. Esto es obligatorio.

```
# CORRECTO
if 5 > 3:
    print("5 es mayor que 3")
    print("Esta línea está dentro del if")

print("Esta línea está fuera del if")

# INCORRECTO (genera error)
# if 5 > 3:
# print("Esto generaría IndentationError")
```

Convenciones de estilo (PEP 8)

```python
# ✅ NOMBRES CLAROS Y DESCRIPTIVOS
nombre_usuario = "Juan"
edad = 25
saldo_cuenta = 1000.50

# ❌ EVITAR
nu = "Juan"
e = 25
s = 1000.50

# ✅ ESPACIOS EN OPERADORES
resultado = (x + 5) * (y - 3)

# ❌ SIN ESPACIOS
resultado=(x+5)*(y-3)

# ✅ LÍNEAS CON MÁXIMO 79 CARACTERES
mensaje = ("Este es un mensaje largo que se divide "
           "en múltiples líneas para mantener legibilidad")
```

Palabras Reservadas

Son palabras con significado especial que Python usa internamente:

```python
# Palabras clave de Python
# if, else, elif, for, while, break, continue
# def, class, return, import, from, as
# try, except, finally, raise, with
# and, or, not, in, is
# True, False, None

# No puedes usar estas como nombres de variables
# if = 10   ❌ SyntaxError
# class = "Juan"   ❌ SyntaxError
```

Punto de entrada del programa

```python
# OPCIÓN 1: Ejecución directa
print("Este código siempre se ejecuta")

# OPCIÓN 2: Usar if __name__ == "__main__" (buena práctica)
def saludar(nombre):
    print(f"¡Hola, {nombre}!")

# Este bloque SOLO se ejecuta si el archivo se ejecuta directamente
if __name__ == "__main__":
    saludar("Maria")
    print("Programa iniciado")
```

Ejercicio 1.1 - Básico:

```
# EJERCICIO: Crea un programa que muestre tu información
# Requerimientos:
# - Mostrar tu nombre
# - Mostrar tu edad
# - Mostrar tu ciudad
# - Incluir comentarios

# SOLUCIÓN
# Mi información personal
nombre = "Carlos"
edad = 28
ciudad = "Madrid"

print(f"Nombre: {nombre}")
print(f"Edad: {edad}")
print(f"Ciudad: {ciudad}")
```

2️⃣ Variables y Tipos de Datos

Declaración y Asignación

```
# Python no requiere declarar el tipo (tipado dinámico)
x = 10           # Automáticamente es int
y = 3.14         # Automáticamente es float
nombre = "Ana"   # Automáticamente es str
activo = True    # Automáticamente es bool

# Asignación múltiple
a, b, c = 1, 2, 3
print(a, b, c)  # 1 2 3

# Asignación múltiple con el mismo valor
x = y = z = 0
print(x, y, z)  # 0 0 0
```

Tipado Dinámico vs Estático

```
# TIPADO DINÁMICO (Python)
valor = 5          # int
print(type(valor))  # <class 'int'>

valor = "Hola"     # str (cambió el tipo)
print(type(valor))  # <class 'str'>

valor = 3.14       # float
```

```
print(type(valor))  # <class 'float'>

# TIPADO ESTÁTICO (Anotaciones de tipo - Python 3.5+)
# No es obligatorio, pero mejora la documentación
def suma(a: int, b: int) -> int:
    return a + b

resultado: int = suma(5, 3)
print(resultado)  # 8
```

Tipos Primitivos

Números (int, float):

```
# ENTEROS (int)
entero1 = 42
entero2 = -15
entero3 = 0

# FLOTANTES (float)
flotante1 = 3.14
flotante2 = -2.5
flotante3 = 1.0

# Números en notación científica
notacion_cientifica = 2.5e3  # 2500.0
exponente_negativo = 1e-2    # 0.01

# Operaciones con números
print(10 + 3)      # 13 (suma)
print(10 - 3)      # 7 (resta)
print(10 * 3)      # 30 (multiplicación)
print(10 / 3)      # 3.3333... (división decimal)
print(10 // 3)     # 3 (división entera)
print(10 % 3)      # 1 (módulo)
print(2 ** 3)      # 8 (potencia)
```

Booleanos:

```
verdadero = True
falso = False

# Comparaciones devuelven booleanos
print(5 > 3)       # True
print(5 == 3)      # False
print(5 != 3)      # True

# Operaciones lógicas
print(True and False)   # False
```

```
print(True or False)    # True
print(not True)         # False
```

Strings (Cadenas de texto):

```
# Diferentes formas de declarar strings
string1 = "Hola"
string2 = 'Mundo'
string3 = """Este es un
string de múltiples
líneas"""

# Concatenación
saludo = "Hola" + " " + "Mundo"
print(saludo)  # Hola Mundo

# Repetición
print("*" * 5)  # *****

# Acceso a caracteres (indexación)
palabra = "Python"
print(palabra[0])   # P (primer carácter)
print(palabra[-1])  # n (último carácter)

# Slicing (troceado)
print(palabra[0:3])   # Pyt (caracteres 0 a 2)
print(palabra[2:])    # thon (desde índice 2 al final)
print(palabra[:3])    # Pyt (desde el inicio hasta índice 2)

# Métodos de strings
print(palabra.lower())        # python
print(palabra.upper())        # PYTHON
print(palabra.replace("P", "J"))  # Jython
print("  espacios  ".strip())  # espacios (sin espacios al inicio/final)
print(palabra.startswith("Py"))   # True
print(palabra.endswith("on"))     # True
```

None (Null de Python):

```
# None representa la ausencia de valor
valor_nulo = None
print(valor_nulo)  # None
print(type(valor_nulo))  # <class 'NoneType'>

# Comprobación de None
if valor_nulo is None:
    print("El valor es None")

# Diferencia entre None, False, 0 y ""
print(None == False)   # False
```

```
print(None == 0)       # False
print(None == "")      # False
print(None is None)    # True (la forma correcta de comparar)
```

Conversión de Tipos (Casting)

```
# De string a número
numero_str = "42"
numero_int = int(numero_str)
print(numero_int + 8)  # 50

# De número a string
numero = 42
numero_str = str(numero)
print(numero_str + " es la respuesta")  # 42 es la respuesta

# De float a int (trunca decimales)
flotante = 3.99
entero = int(flotante)
print(entero)  # 3

# De int a float
entero = 5
flotante = float(entero)
print(flotante)  # 5.0

# A booleano
print(bool(0))      # False
print(bool(1))      # True
print(bool(""))     # False
print(bool("Hola")) # True
print(bool([]))     # False
print(bool([1, 2])) # True

# Manejo de errores en conversión
try:
    numero = int("abc")  # Esto causará un error
except ValueError:
    print("No se puede convertir 'abc' a número")
```

Ejercicio 2.1 - Tipos de datos:

```
# EJERCICIO: Crea un programa calculador de IMC
# Fórmula: IMC = peso (kg) / altura² (m)
# Requerimientos:
# - Solicitar peso (número)
# - Solicitar altura (número)
# - Calcular IMC
# - Mostrar resultado
```

```python
# SOLUCIÓN
print("=== CALCULADORA DE IMC ===")
peso = float(input("Ingresa tu peso en kg: "))
altura = float(input("Ingresa tu altura en metros: "))

imc = peso / (altura ** 2)
print(f"Tu IMC es: {imc:.2f}")

# Clasificación
if imc < 18.5:
    print("Bajo peso")
elif imc < 25:
    print("Peso normal")
elif imc < 30:
    print("Sobrepeso")
else:
    print("Obesidad")
```

3️⃣ Operadores

Operadores Aritméticos

```python
a = 10
b = 3

print(a + b)   # 13 (suma)
print(a - b)   # 7 (resta)
print(a * b)   # 30 (multiplicación)
print(a / b)   # 3.333... (división decimal)
print(a // b)  # 3 (división entera)
print(a % b)   # 1 (módulo - residuo)
print(a ** b)  # 1000 (potencia)

# Operador // y % con negativos
print(-10 // 3)   # -4
print(-10 % 3)    # 2
```

Operadores Relacionales (Comparación)

```python
x = 10
y = 20

print(x == y)   # False (igualdad)
print(x != y)   # True (desigualdad)
print(x < y)    # True (menor que)
print(x > y)    # False (mayor que)
print(x <= y)   # True (menor o igual)
```

```
print(x >= y)   # False (mayor o igual)

# Comparaciones encadenadas
print(5 < 10 < 15)    # True
print(10 >= 10 > 5)   # True
print(1 < 2 < 3 < 2)  # False
```

Operadores Lógicos

```
# AND (y)
print(True and True)    # True
print(True and False)   # False
print(False and False)  # False

# OR (o)
print(True or False)    # True
print(False or False)   # False

# NOT (no)
print(not True)         # False
print(not False)        # True

# Ejemplo práctico
edad = 25
tiene_licencia = True

if edad >= 18 and tiene_licencia:
    print("Puedes conducir")

if edad < 18 or not tiene_licencia:
    print("No puedes conducir")
```

Operadores de Asignación

```
x = 10  # Asignación simple

# Asignación compuesta
x += 5      # x = x + 5 → 15
x -= 3      # x = x - 3 → 12
x *= 2      # x = x * 2 → 24
x /= 4      # x = x / 4 → 6.0
x //= 2     # x = x // 2 → 3.0
x %= 2      # x = x % 2 → 1.0
x **= 3     # x = x ** 3 → 1.0

# Operador de asignación walrus (:=) - Python 3.8+
if (n := len("hola")) > 3:
    print(f"La palabra tiene {n} caracteres")
```

Operadores Especiales

```
# Operador 'in' - verifica pertenencia
print('a' in 'hola')      # True
print(3 in [1, 2, 3, 4])  # True
print('z' in 'hola')      # False

# Operador 'is' - verifica identidad
x = [1, 2, 3]
y = x
z = [1, 2, 3]

print(x is y)   # True (mismo objeto en memoria)
print(x is z)   # False (diferentes objetos)
print(x == z)   # True (mismo contenido)

# Operador 'not in'
print('x' not in 'hola')  # True
```

Ejercicio 3.1 - Operadores:

```
# EJERCICIO: Crea un programa que determine si una persona
# puede solicitar un crédito
# Requerimientos:
# - Edad >= 18 años
# - Ingresos >= $2000 mensuales
# - Sin deudas activas
# - Mostrar si es elegible o no

# SOLUCIÓN
print("=== EVALUADOR DE CRÉDITO ===")
edad = int(input("¿Cuántos años tienes? "))
ingresos = float(input("¿Cuál es tu ingreso mensual? "))
tiene_deudas = input("¿Tienes deudas activas? (si/no): ").lower()

tiene_deudas = tiene_deudas == "si"

es_elegible = (edad >= 18 and
               ingresos >= 2000 and
               not tiene_deudas)

if es_elegible:
    print("✓ ¡Eres elegible para solicitar un crédito!")
else:
    print("✗ No cumples los requisitos para el crédito")
```

4️⃣ Estructuras de Control

if / else / elif

```
# IF simple
edad = 20
if edad >= 18:
    print("Eres mayor de edad")

# IF - ELSE
edad = 15
if edad >= 18:
    print("Eres mayor de edad")
else:
    print("Eres menor de edad")

# IF - ELIF - ELSE
edad = 65
if edad < 13:
    print("Eres un niño")
elif edad < 18:
    print("Eres un adolescente")
elif edad < 65:
    print("Eres un adulto")
else:
    print("Eres un adulto mayor")

# Múltiples ELIFs
calificacion = 85
if calificacion >= 90:
    print("A - Excelente")
elif calificacion >= 80:
    print("B - Bueno")
elif calificacion >= 70:
    print("C - Aceptable")
elif calificacion >= 60:
    print("D - Insuficiente")
else:
    print("F - Reprobado")
```

Operador Ternario

```
# Sintaxis: valor_si_verdadero if condición else valor_si_falso
edad = 25
estado = "Mayor de edad" if edad >= 18 else "Menor de edad"
print(estado)

# Ternario anidado
nota = 85
resultado = "Excelente" if nota >= 90 else "Bueno" if nota >= 80 else "Aprobado" if nota >= 60
print(resultado)  # Bueno
```

```python
# Ternario en asignación de variables
temperatura = 30
clima = "Caluroso" if temperatura > 25 else "Fresco"
print(clima)

# Ternario con funciones
def obtener_descuento(monto):
    return 10 if monto > 100 else 5
```

Switch / Match (Python 3.10+)

Python no tiene `switch` tradicional, pero desde Python 3.10 tiene `match`:

```python
# PRE-PYTHON 3.10: Usar diccionario
def clasificar_animal(animal):
    animales = {
        "perro": "Mamífero",
        "gato": "Mamífero",
        "águila": "Ave",
        "cocodrilo": "Reptil",
    }
    return animales.get(animal, "Desconocido")

print(clasificar_animal("perro"))    # Mamífero
print(clasificar_animal("loro"))     # Desconocido

# PYTHON 3.10+: Match/Case
def clasificar_dia(numero):
    match numero:
        case 1:
            return "Lunes"
        case 2:
            return "Martes"
        case 3:
            return "Miércoles"
        case 4:
            return "Jueves"
        case 5:
            return "Viernes"
        case 6:
            return "Sábado"
        case 7:
            return "Domingo"
        case _:  # Default
            return "Número inválido"

print(clasificar_dia(3))  # Miércoles
```

Ejercicio 4.1 - Estructuras de control:

```python
# EJERCICIO: Crea un programa de cálculo de tarifas de taxi
# Tarifa base: $5
# Por km: $2
# Hora pico (18:00-22:00): +50% en el total
# Mostrar el precio final

# SOLUCIÓN
print("=== CALCULADORA DE TARIFA DE TAXI ===")
distancia = float(input("¿Cuántos km viajarás? "))
hora = int(input("¿A qué hora? (0-23): "))

tarifa_base = 5
precio_por_km = 2

precio = tarifa_base + (distancia * precio_por_km)

# Aplicar recargo de hora pico
if 18 <= hora < 22:
    precio *= 1.50  # +50%
    print("Hora pico: +50%")

print(f"Precio final: ${precio:.2f}")
```

5️⃣ Bucles

Bucle for

```python
# FOR con range
for i in range(5):
    print(i)  # 0, 1, 2, 3, 4

# FOR con range personalizado
for i in range(1, 6):  # inicio, fin (no incluido)
    print(i)  # 1, 2, 3, 4, 5

# FOR con step (incremento)
for i in range(0, 10, 2):
    print(i)  # 0, 2, 4, 6, 8

# FOR para recorrer strings
palabra = "Python"
for letra in palabra:
    print(letra)  # P, y, t, h, o, n

# FOR para recorrer listas
numeros = [10, 20, 30, 40]
for numero in numeros:
    print(numero)
```

```python
# FOR con enumerate (obtener índice)
frutas = ["manzana", "plátano", "naranja"]
for indice, fruta in enumerate(frutas):
    print(f"{indice}: {fruta}")
# 0: manzana
# 1: plátano
# 2: naranja

# FOR anidado
for i in range(3):
    for j in range(3):
        print(f"({i}, {j})", end=" ")  # (0, 0) (0, 1) (0, 2) (1, 0)...

# FOR con else (se ejecuta si el bucle NO se interrumpe)
for i in range(5):
    if i == 10:
        break
else:
    print("El bucle completó sin interrupciones")  # Se muestra
```

Bucle while

```python
# WHILE simple
contador = 0
while contador < 5:
    print(contador)
    contador += 1  # 0, 1, 2, 3, 4

# WHILE con condición compleja
usuario_correcto = "admin"
contraseña_correcta = "1234"
intentos = 0
max_intentos = 3

while intentos < max_intentos:
    usuario = input("Usuario: ")
    contraseña = input("Contraseña: ")

    if usuario == usuario_correcto and contraseña == contraseña_correcta:
        print("¡Acceso concedido!")
        break
    else:
        intentos += 1
        print(f"Credenciales inválidas. Intentos restantes: {max_intentos - intentos}")

# WHILE con else
num = 0
while num < 5:
    print(num)
    num += 1
```

```
else:
    print("El bucle finalizó normalmente")  # Se muestra
```

Break y Continue

```
# BREAK - termina el bucle
for i in range(10):
    if i == 5:
        break  # Detiene el bucle aquí
    print(i)  # 0, 1, 2, 3, 4

# CONTINUE - salta a la siguiente iteración
for i in range(5):
    if i == 2:
        continue  # Salta cuando i = 2
    print(i)  # 0, 1, 3, 4

# Ejemplo: encontrar un número
numeros = [2, 4, 6, 8, 10, 15, 20]
objetivo = 15

for numero in numeros:
    if numero == objetivo:
        print(f"Encontré el {objetivo}")
        break
    else:
        print(f"Revisando {numero}")

# Ejemplo: filtrar números
for i in range(10):
    if i % 2 == 0:  # Si es par
        continue  # Salta
    print(i)  # 1, 3, 5, 7, 9 (impares)
```

Do-While (Python NO tiene, pero se puede simular)

```
# Python NO tiene do-while nativo
# Se simula con while True y break

# Simulación de do-while
while True:
    numero = int(input("Ingresa un número entre 1 y 10: "))
    if 1 <= numero <= 10:
        print(f"Número válido: {numero}")
        break
    else:
        print("Intenta de nuevo")
```

Ejercicio 5.1 - Bucles:

```python
# EJERCICIO: Crea un programa de tabla de multiplicar
# Requerimientos:
# - Solicitar un número
# - Mostrar su tabla de multiplicar (del 1 al 10)
# - Calcular el total de la suma de resultados

# SOLUCIÓN
print("=== TABLA DE MULTIPLICAR ===")
numero = int(input("¿De qué número quieres la tabla? "))

print(f"\nTabla del {numero}:")
suma_total = 0

for i in range(1, 11):
    resultado = numero * i
    print(f"{numero} x {i} = {resultado}")
    suma_total += resultado

print(f"\nSuma de todos los resultados: {suma_total}")
```

Ejercicio 5.2 - Bucles anidados:

```python
# EJERCICIO: Crea un patrón de pirámide
# Ejemplo: si n=5
#     *
#    * *
#   * * *
#  * * * *
# * * * * *

# SOLUCIÓN
n = 5

for i in range(1, n + 1):
    # Espacios
    for j in range(n - i):
        print(" ", end="")

    # Asteriscos
    for j in range(i):
        print("*", end=" ")

    print()  # Nueva línea
```

6️⃣ Funciones

Declaración y Estructura

```python
# FUNCIÓN BÁSICA
def saludar():
    print("¡Hola, mundo!")

saludar()  # Llamar la función

# FUNCIÓN CON PARÁMETROS
def saludar_persona(nombre):
    print(f"¡Hola, {nombre}!")

saludar_persona("María")

# FUNCIÓN CON MÚLTIPLES PARÁMETROS
def sumar(a, b):
    return a + b

resultado = sumar(5, 3)
print(resultado)  # 8

# FUNCIÓN CON VALOR POR DEFECTO
def presentarse(nombre, edad=0, ciudad="Desconocida"):
    print(f"Nombre: {nombre}")
    print(f"Edad: {edad}")
    print(f"Ciudad: {ciudad}")

presentarse("Juan")                          # Usa valores por defecto
presentarse("María", 25)                     # Personaliza algunos
presentarse("Pedro", 30, "Barcelona")        # Personaliza todos
```

Parámetros

```python
# PARÁMETROS POSICIONALES
def restar(a, b):
    return a - b

print(restar(10, 3))  # 7 (10 - 3)

# ARGUMENTOS CON NOMBRE (keyword arguments)
print(restar(b=3, a=10))  # 7 (el orden no importa)

# MEZCLAR POSICIONALES Y NOMBRADOS
def crear_usuario(nombre, edad, ciudad="Madrid", activo=True):
    return f"{nombre}, {edad} años, {ciudad}, {'Activo' if activo else 'Inactivo'}"

print(crear_usuario("Juan", 25))
print(crear_usuario("María", 30, "Barcelona", False))

# *args - número variable de argumentos posicionales
def sumar_varios(*numeros):
    total = 0
```

```
    for num in numeros:
        total += num
    return total

print(sumar_varios(1, 2, 3))           # 6
print(sumar_varios(1, 2, 3, 4, 5))     # 15

# **kwargs - número variable de argumentos nombrados
def mostrar_datos(**datos):
    for clave, valor in datos.items():
        print(f"{clave}: {valor}")

mostrar_datos(nombre="Juan", edad=25, ciudad="Madrid")
# nombre: Juan
# edad: 25
# ciudad: Madrid

# Combinación
def descripcion(nombre, *args, **kwargs):
    print(f"Nombre: {nombre}")
    print(f"Argumentos: {args}")
    print(f"Datos adicionales: {kwargs}")

descripcion("Juan", 1, 2, 3, ciudad="Madrid", profesión="Ingeniero")
```

Valores de Retorno

```
# RETORNO SIMPLE
def duplicar(numero):
    return numero * 2

print(duplicar(5))  # 10

# RETORNO MÚLTIPLE (tupla)
def obtener_coordenadas():
    return 10, 20  # Devuelve una tupla

x, y = obtener_coordenadas()
print(f"X: {x}, Y: {y}")

# RETORNO CON EARLY EXIT
def validar_edad(edad):
    if edad < 0:
        return "Edad no puede ser negativa"
    if edad < 18:
        return "Eres menor de edad"
    if edad > 100:
        return "Edad sospechosa"
    return "Edad válida"

print(validar_edad(25))   # Edad válida
```

```
print(validar_edad(-5))   # Edad no puede ser negativa

# RETORNO NONE (implícito)
def no_retorna():
    print("No devuelvo nada")
    # Si no hay return, la función retorna None

resultado = no_retorna()
print(resultado)  # None
```

Scope (Alcance de Variables)

```
# VARIABLE GLOBAL
contador_global = 0

def incrementar_global():
    global contador_global  # Modificar variable global
    contador_global += 1

incrementar_global()
print(contador_global)  # 1

# VARIABLE LOCAL
def crear_variable_local():
    variable_local = 42
    print(variable_local)  # 42

crear_variable_local()
# print(variable_local)  # ❌ Error: variable_local no existe aquí

# SCOPE ANIDADO
def funcion_externa():
    x = "externa"

    def funcion_interna():
        x = "interna"  # Crea nueva variable local
        print(x)

    funcion_interna()
    print(x)

funcion_externa()
# interna
# externa

# NONLOCAL (modificar variable del scope padre)
def funcion_externa2():
    x = 10

    def funcion_interna2():
        nonlocal x
```

```
        x = 20  # Modifica la x de la función externa

    funcion_interna2()
    print(x)

funcion_externa2()  # 20
```

Funciones Puras vs Efectos Secundarios

```
# FUNCIÓN PURA (determinista, sin efectos secundarios)
def sumar_puro(a, b):
    return a + b

# Siempre devuelve el mismo resultado con los mismos argumentos
print(sumar_puro(5, 3))  # 8
print(sumar_puro(5, 3))  # 8

# FUNCIÓN CON EFECTOS SECUNDARIOS
contador = 0

def incrementar_contador():
    global contador
    contador += 1
    print(f"Contador: {contador}")
    return contador

# Cada llamada modifica el estado externo
incrementar_contador()  # Contador: 1
incrementar_contador()  # Contador: 2

# FUNCIÓN QUE MODIFICA DATOS (no es pura)
def agregar_item_impuro(lista, item):
    lista.append(item)  # Modifica la lista original
    return lista

mi_lista = [1, 2, 3]
resultado = agregar_item_impuro(mi_lista, 4)
print(mi_lista)  # [1, 2, 3, 4] - lista original modificada

# FUNCIÓN PURA (no modifica el original)
def agregar_item_puro(lista, item):
    nueva_lista = lista + [item]  # Crea nueva lista
    return nueva_lista

otra_lista = [1, 2, 3]
resultado = agregar_item_puro(otra_lista, 4)
print(otra_lista)  # [1, 2, 3] - original sin cambios
print(resultado)   # [1, 2, 3, 4]
```

Ejercicio 6.1 - Funciones básicas:

```python
# EJERCICIO: Crear funciones para manejar calificaciones
# - calcular_promedio: recibe lista de calificaciones
# - obtener_letra: recibe número y devuelve letra (A, B, C, D, F)
# - mostrar_reporte: muestra nombre, calificaciones, promedio y letra

# SOLUCIÓN
def calcular_promedio(calificaciones):
    return sum(calificaciones) / len(calificaciones)

def obtener_letra(promedio):
    if promedio >= 90:
        return "A"
    elif promedio >= 80:
        return "B"
    elif promedio >= 70:
        return "C"
    elif promedio >= 60:
        return "D"
    else:
        return "F"

def mostrar_reporte(nombre, calificaciones):
    promedio = calcular_promedio(calificaciones)
    letra = obtener_letra(promedio)

    print(f"\n=== REPORTE DE {nombre.upper()} ===")
    print(f"Calificaciones: {calificaciones}")
    print(f"Promedio: {promedio:.2f}")
    print(f"Letra: {letra}")

# Uso
mostrar_reporte("Juan", [85, 90, 78, 92])
mostrar_reporte("María", [95, 98, 94, 96])
```

Ejercicio 6.2 - Funciones avanzadas:

```python
# EJERCICIO: Crear una calculadora con funciones
# - suma, resta, multiplicación, división
# - Usar *args para sumar múltiples números
# - Usar **kwargs para operaciones especiales

# SOLUCIÓN
def suma(*numeros):
    return sum(numeros)

def resta(a, b):
    return a - b

def multiplicacion(*numeros):
    resultado = 1
```

```
    for num in numeros:
        resultado *= num
    return resultado

def division(a, b):
    if b == 0:
        return "Error: división por cero"
    return a / b

def calculadora(operacion, **valores):
    if operacion == "suma":
        return suma(*valores.get("numeros", []))
    elif operacion == "resta":
        return resta(valores["a"], valores["b"])
    elif operacion == "multiplicacion":
        return multiplicacion(*valores.get("numeros", []))
    elif operacion == "division":
        return division(valores["a"], valores["b"])

# Uso
print(suma(1, 2, 3, 4, 5))  # 15
print(multiplicacion(2, 3, 4))  # 24
print(division(10, 2))  # 5.0
```

🧩 NIVEL INTERMEDIO (Competencia Real)

7️⃣ Estructuras de Datos

Listas (Arrays)

```
# CREACIÓN DE LISTAS
lista_vacia = []
numeros = [1, 2, 3, 4, 5]
mixta = [1, "hola", 3.14, True, None]

# ACCESO A ELEMENTOS
print(numeros[0])      # 1 (primer elemento)
print(numeros[-1])     # 5 (último elemento)
print(numeros[1:3])    # [2, 3] (slicing)

# MODIFICACIÓN
numeros[0] = 10
print(numeros)  # [10, 2, 3, 4, 5]

# MÉTODOS DE LISTA
lista = [3, 1, 4, 1, 5]

lista.append(9)        # Agregar al final: [3, 1, 4, 1, 5, 9]
```

```
lista.insert(0, 0)     # Insertar en posición: [0, 3, 1, 4, 1, 5, 9]
lista.remove(1)        # Remover primer 1: [0, 3, 4, 1, 5, 9]
elemento = lista.pop() # Quita y devuelve último: 9
elemento = lista.pop(0) # Quita en posición 0: 0

lista.clear()          # Vaciar lista
nueva_lista = [3, 1, 4]
nueva_lista.sort()     # Ordenar: [1, 3, 4]
nueva_lista.reverse()  # Invertir: [4, 3, 1]

# BÚSQUEDA
numeros = [10, 20, 30, 40]
print(30 in numeros)   # True
print(numeros.index(30)) # 2 (índice de 30)
print(numeros.count(10))  # 1 (cuántas veces aparece 10)

# SLICING AVANZADO
lista = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
print(lista[2:7])      # [2, 3, 4, 5, 6]
print(lista[::2])      # [0, 2, 4, 6, 8] (cada 2)
print(lista[::-1])     # [9, 8, 7, ..., 1, 0] (inverso)

# LIST COMPREHENSION
cuadrados = [x**2 for x in range(5)]
print(cuadrados)       # [0, 1, 4, 9, 16]

pares = [x for x in range(10) if x % 2 == 0]
print(pares)           # [0, 2, 4, 6, 8]

# COPIAR LISTA
lista_original = [1, 2, 3]
lista_copia_superficial = lista_original[:]  # o .copy()
lista_copia_profunda = lista_original.copy()

lista_copia_superficial.append(4)
print(lista_original)  # [1, 2, 3] (no cambia)
```

Tuplas

```
# TUPLA - Inmutable (no se puede modificar)
tupla = (1, 2, 3, 4, 5)
print(tupla[0])        # 1
print(tupla[-1])       # 5
print(tupla[1:3])      # (2, 3)

# TUPLA CON UN ELEMENTO
tupla_uno = (42,)      # Coma es obligatoria
tupla_falsa = (42)     # Esto NO es tupla, es int

# TUPLA VACÍA
tupla_vacia = ()
```

```python
# MÉTODOS DE TUPLA
tupla = (1, 2, 2, 3, 3, 3)
print(tupla.count(3))   # 3 (cuántas veces aparece 3)
print(tupla.index(2))   # 1 (índice de primer 2)

# DESEMPAQUETAMIENTO
x, y, z = (10, 20, 30)
print(x, y, z)  # 10 20 30

# INTERCAMBIAR VARIABLES
a, b = 5, 10
a, b = b, a  # Intercambio simple
print(a, b)  # 10 5

# RETORNAR MÚLTIPLES VALORES
def obtener_datos():
    return "Juan", 25, "Madrid"

nombre, edad, ciudad = obtener_datos()
print(nombre, edad, ciudad)
```

Diccionarios (Maps)

```python
# CREACIÓN
diccionario_vacio = {}
persona = {
    "nombre": "Juan",
    "edad": 25,
    "ciudad": "Madrid",
    "email": "juan@example.com"
}

# ACCESO
print(persona["nombre"])        # Juan
print(persona.get("edad"))      # 25
print(persona.get("teléfono"))  # None (no lanza error)
print(persona.get("teléfono", "No disponible"))  # No disponible

# MODIFICACIÓN
persona["edad"] = 26
persona["teléfono"] = "123456789"
print(persona)

# MÉTODOS
print(persona.keys())      # dict_keys(['nombre', 'edad', 'ciudad', ...])
print(persona.values())    # dict_values(['Juan', 26, 'Madrid', ...])
print(persona.items())     # Tuplas (clave, valor)

# ELIMINAR
del persona["teléfono"]
```

```
persona.pop("email")       # Elimina y devuelve el valor
persona.popitem()          # Elimina el último elemento

# VERIFICAR EXISTENCIA
print("nombre" in persona)     # True
print("teléfono" in persona)   # False

# RECORRER
for clave in persona:
    print(f"{clave}: {persona[clave]}")

for clave, valor in persona.items():
    print(f"{clave}: {valor}")

# ACTUALIZAR
persona.update({"edad": 27, "país": "España"})

# DICCIONARIO ANIDADO
usuarios = {
    "usuario1": {
        "nombre": "Juan",
        "edad": 25
    },
    "usuario2": {
        "nombre": "María",
        "edad": 30
    }
}

print(usuarios["usuario1"]["nombre"])  # Juan

# DICCIONARIO COMPREHENSION
cuadrados_dict = {x: x**2 for x in range(5)}
print(cuadrados_dict)  # {0: 0, 1: 1, 2: 4, 3: 9, 4: 16}
```

Sets (Conjuntos)

```
# CREACIÓN
conjunto_vacio = set()  # Nota: {} crea un diccionario vacío
numeros = {1, 2, 3, 4, 5}
letras = set("hola")    # {'h', 'o', 'l', 'a'}

# CARACTERÍSTICA: No contiene duplicados
numeros_con_duplicados = {1, 2, 2, 3, 3, 3}
print(numeros_con_duplicados)  # {1, 2, 3}

# AGREGAR ELEMENTOS
conjunto = {1, 2, 3}
conjunto.add(4)
print(conjunto)  # {1, 2, 3, 4}
```

```
# AGREGAR MÚLTIPLES
conjunto.update([5, 6, 7])
print(conjunto)  # {1, 2, 3, 4, 5, 6, 7}

# REMOVER ELEMENTOS
conjunto.remove(1)     # Lanza error si no existe
conjunto.discard(2)    # No lanza error si no existe

# OPERACIONES DE CONJUNTOS
a = {1, 2, 3, 4}
b = {3, 4, 5, 6}

union = a | b              # {1, 2, 3, 4, 5, 6}
interseccion = a & b       # {3, 4}
diferencia = a - b         # {1, 2}
diferencia_simetrica = a ^ b  # {1, 2, 5, 6}

print(union)
print(interseccion)
print(diferencia)

# MÉTODOS
print(a.union(b))          # {1, 2, 3, 4, 5, 6}
print(a.intersection(b))   # {3, 4}
print(a.difference(b))     # {1, 2}
print(a.symmetric_difference(b))  # {1, 2, 5, 6}

# COMPARACIÓN
print(a.issubset(b))       # False (¿a es subconjunto de b?)
print(a.issuperset(b))     # False (¿a es superconjunto de b?)
print(a.isdisjoint(b))     # False (¿no tienen elementos en común?)
```

Ejercicio 7.1 - Estructuras de datos:

```
# EJERCICIO: Crea un sistema de gestión de contactos
# - Agregar contactos (nombre, teléfono, email)
# - Buscar contacto por nombre
# - Mostrar todos los contactos
# - Eliminar contacto
# - Usar diccionario principal

# SOLUCIÓN
contactos = {}

def agregar_contacto(nombre, telefono, email):
    if nombre in contactos:
        print(f"El contacto {nombre} ya existe")
    else:
        contactos[nombre] = {
            "telefono": telefono,
            "email": email
```

```
        }
        print(f"Contacto {nombre} agregado")

def buscar_contacto(nombre):
    if nombre in contactos:
        print(f"\n=== {nombre.upper()} ===")
        print(f"Teléfono: {contactos[nombre]['telefono']}")
        print(f"Email: {contactos[nombre]['email']}")
    else:
        print(f"Contacto {nombre} no encontrado")

def mostrar_todos():
    if not contactos:
        print("No hay contactos")
    else:
        print("\n=== CONTACTOS ===")
        for nombre, datos in contactos.items():
            print(f"{nombre}: {datos['telefono']} - {datos['email']}")

def eliminar_contacto(nombre):
    if nombre in contactos:
        del contactos[nombre]
        print(f"Contacto {nombre} eliminado")
    else:
        print(f"Contacto {nombre} no encontrado")

# USO
agregar_contacto("Juan", "123456789", "juan@email.com")
agregar_contacto("María", "987654321", "maria@email.com")
mostrar_todos()
buscar_contacto("Juan")
eliminar_contacto("María")
mostrar_todos()
```

8️⃣ Manejo de Errores

Try / Except / Finally

```
# TRY - EXCEPT BÁSICO
try:
    numero = int("abc")  # Esto causará ValueError
except ValueError:
    print("Error: No se puede convertir a número")

# MÚLTIPLES EXCEPCIONES
try:
    numero = int(input("Ingresa un número: "))
    resultado = 10 / numero
except ValueError:
```

```
    print("Error: Debes ingresar un número")
except ZeroDivisionError:
    print("Error: No puedes dividir entre cero")

# CAPTURAR CUALQUIER EXCEPCIÓN
try:
    lista = [1, 2, 3]
    print(lista[10])  # IndexError
except Exception as e:
    print(f"Ocurrió un error: {e}")

# TRY - EXCEPT - ELSE
try:
    numero = int(input("Número: "))
    resultado = 10 / numero
except ValueError:
    print("Debe ser un número")
except ZeroDivisionError:
    print("No puedes dividir entre cero")
else:
    print(f"Resultado: {resultado}")  # Se ejecuta si NO hay error

# TRY - EXCEPT - FINALLY
try:
    archivo = open("datos.txt", "r")
    contenido = archivo.read()
except FileNotFoundError:
    print("Archivo no encontrado")
finally:
    print("Limpieza de recursos")  # SIEMPRE se ejecuta
    # archivo.close()  # Importante: cerrar archivo
```

Throw (Raise) - Lanzar Excepciones

```
# LANZAR EXCEPCIÓN SIMPLE
def verificar_edad(edad):
    if edad < 0:
        raise ValueError("La edad no puede ser negativa")
    if edad < 18:
        raise ValueError("Debes ser mayor de 18 años")
    return "Edad válida"

try:
    print(verificar_edad(25))
except ValueError as error:
    print(f"Error: {error}")

# RELANZAR EXCEPCIÓN
def procesar_datos(datos):
    try:
        numero = int(datos)
```

```
    except ValueError as e:
        print(f"Error al procesar: {e}")
        raise  # Relanza la excepción

try:
    procesar_datos("abc")
except ValueError:
    print("Se capturó la excepción relanzada")

# ENCADENA EXCEPCIONES
try:
    resultado = 10 / 0
except ZeroDivisionError as e:
    raise ValueError("Operación inválida") from e
```

Excepciones Personalizadas

```
# CREAR EXCEPCIÓN PERSONALIZADA
class SaldoInsuficiente(Exception):
    """Excepción cuando no hay saldo suficiente"""
    pass

class CuentaNoExiste(Exception):
    """Excepción cuando la cuenta no existe"""
    pass

# USAR EXCEPCIONES PERSONALIZADAS
class CuentaBancaria:
    def __init__(self, titular, saldo=0):
        self.titular = titular
        self.saldo = saldo

    def retirar(self, cantidad):
        if cantidad > self.saldo:
            raise SaldoInsuficiente(f"Saldo insuficiente. Disponible: ${self.saldo}")

        self.saldo -= cantidad
        print(f"Retiro exitoso. Nuevo saldo: ${self.saldo}")

    def depositar(self, cantidad):
        if cantidad <= 0:
            raise ValueError("La cantidad debe ser positiva")

        self.saldo += cantidad
        print(f"Depósito exitoso. Nuevo saldo: ${self.saldo}")

# USO
cuenta = CuentaBancaria("Juan", 1000)

try:
    cuenta.retirar(500)
```

```
    cuenta.retirar(700)  # Esto lanzará excepción
except SaldoInsuficiente as error:
    print(f"Operación rechazada: {error}")

try:
    cuenta.depositar(-100)
except ValueError as error:
    print(f"Error: {error}")
```

Ejercicio 8.1 - Manejo de errores:

```
# EJERCICIO: Calculadora robusta con manejo de errores
# - Solicitar dos números
# - Solicitar operación (+, -, *, /)
# - Realizar cálculo
# - Manejar todos los errores posibles

# SOLUCIÓN
def calculadora_robusta():
    try:
        numero1 = float(input("Primer número: "))
        numero2 = float(input("Segundo número: "))
        operacion = input("Operación (+, -, *, /): ")

        if operacion == "+":
            resultado = numero1 + numero2
        elif operacion == "-":
            resultado = numero1 - numero2
        elif operacion == "*":
            resultado = numero1 * numero2
        elif operacion == "/":
            if numero2 == 0:
                raise ZeroDivisionError("No puedes dividir entre cero")
            resultado = numero1 / numero2
        else:
            raise ValueError(f"Operación '{operacion}' no válida")

        print(f"Resultado: {resultado}")

    except ValueError as e:
        print(f"Error de valor: {e}")
    except ZeroDivisionError as e:
        print(f"Error matemático: {e}")
    except Exception as e:
        print(f"Error inesperado: {e}")
    finally:
        print("Calculadora cerrada")

calculadora_robusta()
```

9️⃣ Programación Orientada a Objetos (POO)

Clases y Objetos

```
# DEFINIR UNA CLASE
class Persona:
    # Constructor (método especial)
    def __init__(self, nombre, edad):
        self.nombre = nombre
        self.edad = edad

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

    # Método que modifica estado
    def envejecer(self):
        self.edad += 1

# CREAR OBJETOS (instancias)
persona1 = Persona("Juan", 25)
persona2 = Persona("María", 30)

# ACCEDER A ATRIBUTOS
print(persona1.nombre)      # Juan
print(persona1.edad)        # 25

# LLAMAR MÉTODOS
print(persona1.presentarse())  # Hola, soy Juan y tengo 25 años
persona1.envejecer()
print(persona1.edad)        # 26

# TIPO DE OBJETO
print(type(persona1))       # <class '__main__.Persona'>
print(isinstance(persona1, Persona))  # True
```

Atributos de Clase vs Instancia

```
class Contador:
    contador_total = 0  # Atributo de CLASE (compartido)

    def __init__(self, nombre):
        self.nombre = nombre      # Atributo de INSTANCIA (individual)
        Contador.contador_total += 1

    def obtener_total(self):
        return Contador.contador_total

obj1 = Contador("A")
obj2 = Contador("B")
```

```
obj3 = Contador("C")

print(obj1.obtener_total())     # 3
print(Contador.contador_total)  # 3

# Los atributos de instancia son independientes
obj1.nombre = "Z"
print(obj1.nombre)  # Z
print(obj2.nombre)  # B
```

Métodos: Instancia, Clase y Estáticos

```
class Utilidades:
    nombre_clase = "Utilidades"

    # MÉTODO DE INSTANCIA
    def metodo_instancia(self):
        return f"Soy un método de instancia. Clase: {self.nombre_clase}"

    # MÉTODO DE CLASE (recibe la clase, no la instancia)
    @classmethod
    def metodo_clase(cls):
        return f"Soy un método de clase. Nombre: {cls.nombre_clase}"

    # MÉTODO ESTÁTICO (no necesita instancia ni clase)
    @staticmethod
    def metodo_estatico():
        return "Soy un método estático"

obj = Utilidades()
print(obj.metodo_instancia())  # Soy un método de instancia...
print(Utilidades.metodo_clase())  # Soy un método de clase...
print(Utilidades.metodo_estatico())  # Soy un método estático

# Métodos estáticos útiles para utilidades
class Conversor:
    @staticmethod
    def celsius_a_fahrenheit(celsius):
        return (celsius * 9/5) + 32

    @staticmethod
    def fahrenheit_a_celsius(fahrenheit):
        return (fahrenheit - 32) * 5/9

print(Conversor.celsius_a_fahrenheit(0))    # 32.0
print(Conversor.fahrenheit_a_celsius(32))   # 0.0
```

Encapsulamiento

```python
class CuentaBancaria:
    def __init__(self, titular, saldo_inicial):
        self.titular = titular
        self.__saldo = saldo_inicial  # Atributo PRIVADO (__)
        self._id = "1001"              # Atributo PROTEGIDO (_)

    # GETTER (para leer)
    def obtener_saldo(self):
        return self.__saldo

    # SETTER (para escribir)
    def establecer_saldo(self, nuevo_saldo):
        if nuevo_saldo < 0:
            print("El saldo no puede ser negativo")
        else:
            self.__saldo = nuevo_saldo

    # PROPERTY (decorador para acceso como atributo)
    @property
    def saldo(self):
        return self.__saldo

    @saldo.setter
    def saldo(self, cantidad):
        if cantidad >= 0:
            self.__saldo = cantidad

cuenta = CuentaBancaria("Juan", 1000)

# Usando método
print(cuenta.obtener_saldo())  # 1000
cuenta.establecer_saldo(2000)
print(cuenta.obtener_saldo())  # 2000

# Usando property
print(cuenta.saldo)  # 2000
cuenta.saldo = 3000
print(cuenta.saldo)  # 3000

# No se puede acceder directamente a __saldo (name mangling)
# print(cuenta.__saldo)  # ❌ Error
print(cuenta._CuentaBancaria__saldo)  # 3000 (acceso forzado, evitar)
```

Herencia

```python
# CLASE PADRE
class Animal:
    def __init__(self, nombre):
        self.nombre = nombre
```

```
    def hacer_sonido(self):
        return "Sonido genérico"

    def presentarse(self):
        return f"Soy un animal llamado {self.nombre}"

# CLASES HIJAS
class Perro(Animal):
    def hacer_sonido(self):  # SOBRESCRIBIR método
        return "¡Guau guau!"

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

    def ronronear(self):  # Método adicional
        return "Rrr rrr rrr"

# USO
perro = Perro("Rex")
gato = Gato("Mishi")

print(perro.presentarse())  # Soy un animal llamado Rex
print(perro.hacer_sonido())  # ¡Guau guau!

print(gato.presentarse())  # Soy un animal llamado Mishi
print(gato.hacer_sonido())  # ¡Miau!
print(gato.ronronear())  # Rrr rrr rrr

# HERENCIA MÚLTIPLE
class Volador:
    def volar(self):
        return "Estoy volando"

class Pajaro(Animal, Volador):
    def hacer_sonido(self):
        return "¡Pío pío!"

pajaro = Pajaro("Tweety")
print(pajaro.hacer_sonido())  # ¡Pío pío!
print(pajaro.volar())  # Estoy volando
```

Método super()

```
class Vehiculo:
    def __init__(self, marca, modelo):
        self.marca = marca
        self.modelo = modelo

    def describir(self):
        return f"{self.marca} {self.modelo}"
```

```python
class Auto(Vehiculo):
    def __init__(self, marca, modelo, puertas):
        super().__init__(marca, modelo)  # Llamar constructor padre
        self.puertas = puertas

    def describir(self):
        desc_padre = super().describir()
        return f"{desc_padre} con {self.puertas} puertas"

auto = Auto("Toyota", "Corolla", 4)
print(auto.describir())  # Toyota Corolla con 4 puertas
```

Polimorfismo

```python
# El polimorfismo permite que objetos diferentes
# respondan al mismo mensaje de manera diferente

class Figura:
    def calcular_area(self):
        pass

class Cuadrado(Figura):
    def __init__(self, lado):
        self.lado = lado

    def calcular_area(self):
        return self.lado ** 2

class Rectangulo(Figura):
    def __init__(self, largo, ancho):
        self.largo = largo
        self.ancho = ancho

    def calcular_area(self):
        return self.largo * self.ancho

class Circulo(Figura):
    def __init__(self, radio):
        self.radio = radio

    def calcular_area(self):
        return 3.14159 * (self.radio ** 2)

# FUNCIÓN QUE UTILIZA POLIMORFISMO
def calcular_area_total(figuras):
    total = 0
    for figura in figuras:
        total += figura.calcular_area()
    return total
```

```
figuras = [
    Cuadrado(5),
    Rectangulo(4, 6),
    Circulo(3)
]

print(f"Área total: {calcular_area_total(figuras):.2f}")
```

Abstracción (ABC - Abstract Base Classes)

```
from abc import ABC, abstractmethod

class Animal(ABC):  # Clase abstracta
    @abstractmethod
    def hacer_sonido(self):
        pass

    @abstractmethod
    def moverse(self):
        pass

    def respirar(self):  # Método concreto
        return "Respirando..."

# No puedo instanciar Animal directamente
# animal = Animal()  # ❌ TypeError

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

    def moverse(self):
        return "Corriendo con 4 patas"

class Pajaro(Animal):
    def hacer_sonido(self):
        return "¡Pío!"

    def moverse(self):
        return "Volando"

# Sí puedo instanciar las clases concretas
perro = Perro()
pajaro = Pajaro()

print(perro.hacer_sonido())    # ¡Guau!
print(perro.moverse())         # Corriendo con 4 patas
print(pajaro.hacer_sonido())   # ¡Pío!
print(pajaro.moverse())        # Volando
```

Ejercicio 9.1 - POO Completo:

```
# EJERCICIO: Sistema de gestión de estudiantes y cursos
# Requerimientos:
# - Clase Estudiante (nombre, matrícula, notas)
# - Clase Curso (nombre, profesor, estudiantes)
# - Métodos para agregar estudiantes, calcular promedio
# - Usar herencia, encapsulamiento y polimorfismo

# SOLUCIÓN
class Persona:
    def __init__(self, nombre):
        self.nombre = nombre

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

class Estudiante(Persona):
    def __init__(self, nombre, matricula):
        super().__init__(nombre)
        self.matricula = matricula
        self.__notas = []

    def agregar_nota(self, nota):
        if 0 <= nota <= 100:
            self.__notas.append(nota)
        else:
            raise ValueError("La nota debe estar entre 0 y 100")

    def calcular_promedio(self):
        if not self.__notas:
            return 0
        return sum(self.__notas) / len(self.__notas)

    def presentarse(self):
        return f"{super().presentarse()} (Matrícula: {self.matricula})"

class Profesor(Persona):
    def __init__(self, nombre, especialidad):
        super().__init__(nombre)
        self.especialidad = especialidad

class Curso:
    def __init__(self, nombre, profesor):
        self.nombre = nombre
        self.profesor = profesor
        self.estudiantes = []

    def agregar_estudiante(self, estudiante):
        if isinstance(estudiante, Estudiante):
            self.estudiantes.append(estudiante)
```

```
    def mostrar_estudiantes(self):
        print(f"\n=== CURSO: {self.nombre} ===")
        print(f"Profesor: {self.profesor.nombre}")
        print(f"Especialidad: {self.profesor.especialidad}")
        print("Estudiantes:")
        for est in self.estudiantes:
            print(f"  - {est.presentarse()}: Promedio {est.calcular_promedio():.2f}")

# USO
profesor = Profesor("Dr. Juan", "Matemáticas")
curso = Curso("Cálculo I", profesor)

estudiante1 = Estudiante("Ana", "2024001")
estudiante1.agregar_nota(85)
estudiante1.agregar_nota(90)
estudiante1.agregar_nota(88)

estudiante2 = Estudiante("Carlos", "2024002")
estudiante2.agregar_nota(75)
estudiante2.agregar_nota(80)
estudiante2.agregar_nota(78)

curso.agregar_estudiante(estudiante1)
curso.agregar_estudiante(estudiante2)

curso.mostrar_estudiantes()
```

Ejercicio 9.2 - Sistema de Tienda:

```
# EJERCICIO: Sistema de tienda con productos y carrito
# Requerimientos:
# - Clase Producto (nombre, precio, cantidad)
# - Clase Carrito (agregar, remover, calcular total)
# - Manejo de excepciones para cantidad insuficiente

# SOLUCIÓN
class Producto:
    def __init__(self, nombre, precio, cantidad):
        self.nombre = nombre
        self.precio = precio
        self.cantidad = cantidad

    def reducir_cantidad(self, cantidad):
        if cantidad > self.cantidad:
            raise ValueError(f"Cantidad insuficiente. Disponible: {self.cantidad}")
        self.cantidad -= cantidad

class Carrito:
    def __init__(self):
        self.items = []
```

```
    def agregar_producto(self, producto, cantidad):
        for item in self.items:
            if item["producto"].nombre == producto.nombre:
                item["cantidad"] += cantidad
                return

        self.items.append({
            "producto": producto,
            "cantidad": cantidad
        })

    def remover_producto(self, nombre):
        self.items = [item for item in self.items if item["producto"].nombre != nombre]

    def calcular_total(self):
        total = 0
        for item in self.items:
            total += item["producto"].precio * item["cantidad"]
        return total

    def mostrar_carrito(self):
        print("\n=== CARRITO DE COMPRAS ===")
        if not self.items:
            print("Carrito vacío")
        else:
            for item in self.items:
                producto = item["producto"]
                cantidad = item["cantidad"]
                subtotal = producto.precio * cantidad
                print(f"{producto.nombre}: ${producto.precio} x {cantidad} = ${subtotal}")
            print(f"Total: ${self.calcular_total():.2f}")

# USO
tienda_carrito = Carrito()

producto1 = Producto("Laptop", 800, 5)
producto2 = Producto("Mouse", 25, 50)
producto3 = Producto("Teclado", 75, 20)

tienda_carrito.agregar_producto(producto1, 1)
tienda_carrito.agregar_producto(producto2, 2)
tienda_carrito.agregar_producto(producto3, 1)

tienda_carrito.mostrar_carrito()

tienda_carrito.remover_producto("Mouse")
print("\nDespués de remover Mouse:")
tienda_carrito.mostrar_carrito()
```

📋 Resumen de Conceptos Clave

Nivel Básico

- ✅ Sintaxis y convenciones de Python
- ✅ Tipos de datos primitivos
- ✅ Operadores aritméticos, relacionales y lógicos
- ✅ Estructuras de control (if, else, elif)
- ✅ Bucles (for, while, break, continue)
- ✅ Funciones básicas

Nivel Intermedio

- ✅ Estructuras de datos complejas (listas, tuplas, diccionarios, sets)
- ✅ Manejo robusto de errores
- ✅ Programación Orientada a Objetos completa
- ✅ Herencia, polimorfismo y abstracción
- ✅ Encapsulamiento y propiedades