

UNIVERSIDAD PRIVADA DOMINGO SAVIO



Ejercicios de programación II: Actividad 02

02/07/2025 9:02:12

Docente:

Jimmy Nataniel Requena Llorentty

Estudiante:

Franz A. Almanza Galindo

Maria F. Vidaurre Alvarado

Camila Rebeca Mercado Duran

Rebeca Vargas Orellana

Ricardo Maldonado Suarez

Materia:

Programación II – Turno mañana

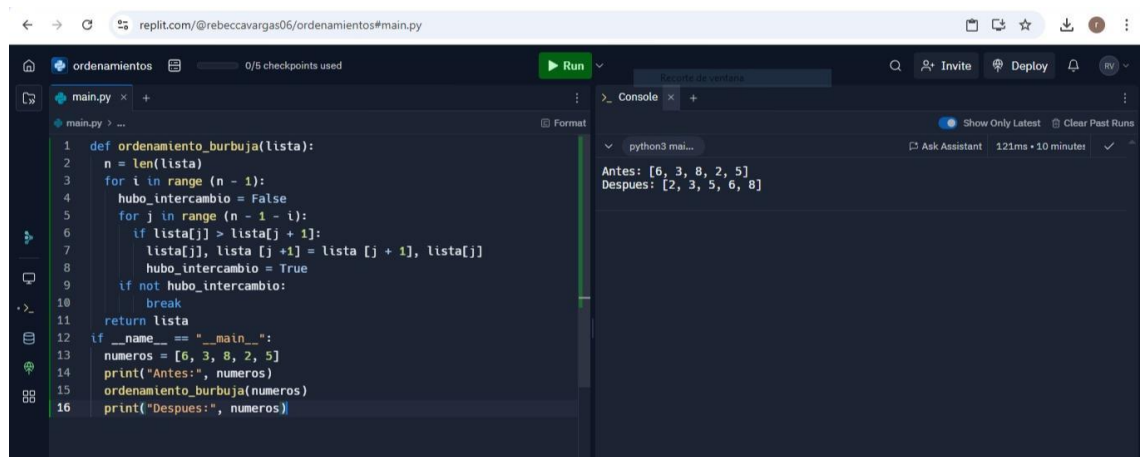
Santa Cruz – Bolivia

Ejercicios de Programación

Actividad 02: Clases de Programación II

```
def ordenamiento_burbuja(lista):
    n = len(lista)
    for i in range(n - 1):
        hubo_intercambio = False
        for j in range(n - 1 - i):
            if lista[j] > lista[j + 1]:
                lista[j], lista[j + 1] = lista[j + 1], lista[j]
                hubo_intercambio = True
        if not hubo_intercambio:
            break
    return lista

if __name__ == "__main__":
    numeros = [6, 3, 8, 2, 5]
    print("Antes:", numeros)
    ordenamiento_burbuja(numeros)
    print("Despues:", numeros)
```

A screenshot of a Replit Python environment. The left pane shows a file named 'main.py' with the bubble sort code. The right pane shows the console output. The code defines a function 'ordenamiento_burbuja' and calls it with the list [6, 3, 8, 2, 5]. The console output shows 'Antes: [6, 3, 8, 2, 5]' and 'Despues: [2, 3, 5, 6, 8]'.

```
ordenamientos 0/5 checkpoints used Run
```

```
main.py x +
main.py > ...
1 def ordenamiento_burbuja(lista):
2     n = len(lista)
3     for i in range(n - 1):
4         hubo_intercambio = False
5         for j in range(n - 1 - i):
6             if lista[j] > lista[j + 1]:
7                 lista[j], lista[j + 1] = lista[j + 1], lista[j]
8                 hubo_intercambio = True
9         if not hubo_intercambio:
10             break
11     return lista
12 if __name__ == "__main__":
13     numeros = [6, 3, 8, 2, 5]
14     print("Antes:", numeros)
15     ordenamiento_burbuja(numeros)
16     print("Despues:", numeros)
```

```
Console x +
python3 mai... Show Only Latest Clear Past Runs
Antes: [6, 3, 8, 2, 5]
Despues: [2, 3, 5, 6, 8]
```

Código: Ordenamiento burbuja

Definición de la función de ordenamiento burbuja

```
def ordenamiento_burbuja(lista):
    n = len(lista)
    for i in range(n):
        for j in range(0, n - i - 1):
            if lista[j] > lista[j + 1]:
                lista[j], lista[j + 1] = lista[j + 1], lista[j]
```

#llamar a la función:

```
lista_a_ordenar = [64, 34, 25, 12, 22, 11, 90]
print(f"Lista original: {lista_a_ordenar}")
```

```
ordenamiento_burbuja(lista_a_ordenar) # Llamamos a la función
print(f"Lista ordenada: {lista_a_ordenar}")
```

Prueba con assert:

Caso 1: Lista desordenada

```
lista1 = [6, 3, 8, 2, 5]
ordenamiento_burbuja(lista1)
assert lista1 == [2, 3, 5, 6, 8], "Falló en Caso 1"
```

Caso 2: Lista ya ordenada

```
lista2 = [1, 2, 3, 4, 5]
ordenamiento_burbuja(lista2)
assert lista2 == [1, 2, 3, 4, 5], "Falló en Caso 2"
```

Caso 3: Lista ordenada a la inversa (peor caso)

```
lista3 = [5, 4, 3, 2, 1]
ordenamiento_burbuja(lista3)
assert lista3 == [1, 2, 3, 4, 5], "Falló en Caso 3"
```

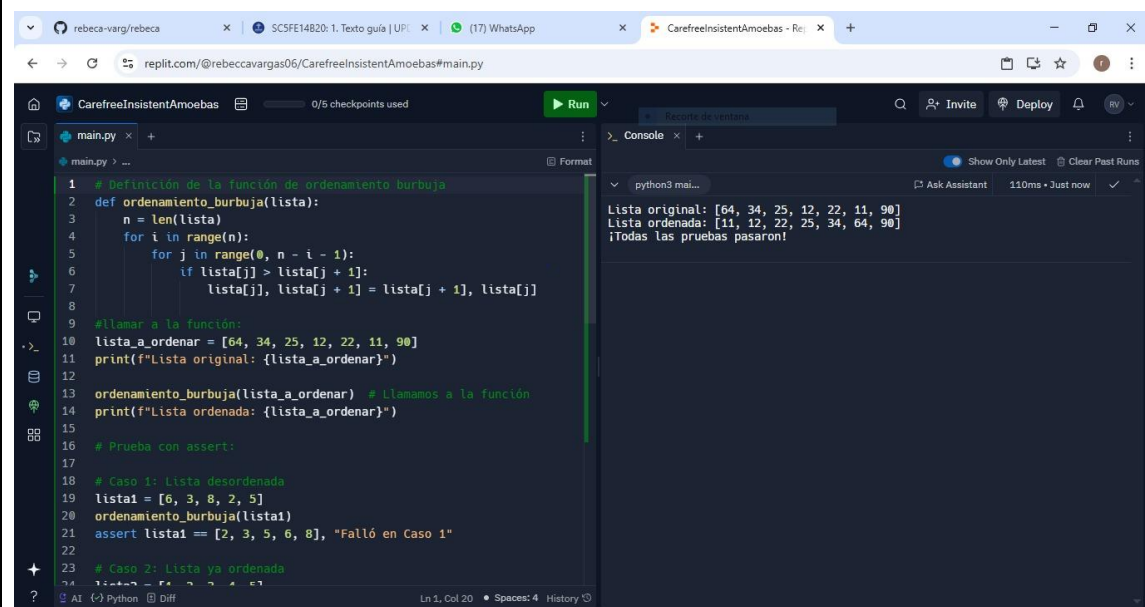
Caso 4: Lista con elementos duplicados

```
lista4 = [5, 1, 4, 2, 5, 5, 2]
ordenamiento_burbuja(lista4)
assert lista4 == [1, 2, 2, 4, 5, 5, 5], "Falló en Caso 4"
```

Casos borde

```
assert ordenamiento_burbuja([]) == None
assert ordenamiento_burbuja([42]) == None, "Fallo en caso borde"
```

```
print("¡Todas las pruebas pasaron!")
```



The screenshot shows a Replit Python environment with a file named `main.py`. The code defines a `ordenamiento_burbuja` function and includes several test cases with `assert` statements. The console output shows the results of running the code, including the original and sorted lists for a specific test case and a confirmation that all tests passed.

```
1 # Definición de la función de ordenamiento burbuja
2 def ordenamiento_burbuja(lista):
3     n = len(lista)
4     for i in range(n):
5         for j in range(0, n - i - 1):
6             if lista[j] > lista[j + 1]:
7                 lista[j], lista[j + 1] = lista[j + 1], lista[j]
8
9 #llamar a la función:
10 lista_a_ordenar = [64, 34, 25, 12, 22, 11, 90]
11 print(f"Lista original: {lista_a_ordenar}")
12
13 ordenamiento_burbuja(lista_a_ordenar) # llamamos a la función
14 print(f"Lista ordenada: {lista_a_ordenar}")
15
16 # Prueba con assert:
17
18 # Caso 1: Lista desordenada
19 lista1 = [6, 3, 8, 2, 5]
20 ordenamiento_burbuja(lista1)
21 assert lista1 == [2, 3, 5, 6, 8], "Falló en Caso 1"
22
23 # Caso 2: Lista ya ordenada
24 lista2 = [1, 2, 3, 4, 5]
25 ordenamiento_burbuja(lista2)
26 assert lista2 == [1, 2, 3, 4, 5], "Falló en Caso 2"
27
28 # Caso 3: Lista ordenada a la inversa (peor caso)
29 lista3 = [5, 4, 3, 2, 1]
30 ordenamiento_burbuja(lista3)
31 assert lista3 == [1, 2, 3, 4, 5], "Falló en Caso 3"
32
33 # Caso 4: Lista con elementos duplicados
34 lista4 = [5, 1, 4, 2, 5, 5, 2]
35 ordenamiento_burbuja(lista4)
36 assert lista4 == [1, 2, 2, 4, 5, 5, 5], "Falló en Caso 4"
37
38 # Casos borde
39 assert ordenamiento_burbuja([]) == None
40 assert ordenamiento_burbuja([42]) == None, "Fallo en caso borde"
41
42 print("¡Todas las pruebas pasaron!")
```

Console Output:

```
python3 mai...
Lista original: [64, 34, 25, 12, 22, 11, 90]
Lista ordenada: [11, 12, 22, 25, 34, 64, 90]
¡Todas las pruebas pasaron!
```

Código: Ordenamiento de menor a mayor

```

matriz= [1, 2, 3], [4, 5, 6], [7, 8, 9]
# OPCION 1: Recorriendo con indices
num_filas = len(matriz)
num_columnas = len(matriz[0])
for i in range(num_filas):
    for j in range(num_columnas):
        elemento = matriz[i][j]
        print(f"elemento en ({i},{j}) es {elemento}")
# OPCION 2: Recorriendo por elemento (mas "pythonico")
for fila_actual in matriz:
    for elemento in fila_actual:
        print(elemento, end="")
    print()

```

```

1 matriz= [1, 2, 3], [4, 5, 6], [7, 8, 9]
2 # OPCION 1: Recorriendo con indices
3 num_filas = len(matriz)
4 num_columnas = len(matriz[0])
5 for i in range(num_filas):
6     for j in range(num_columnas):
7         elemento = matriz[i][j]
8         print(f"elemento en ({i},{j}) es {elemento}")
9 # OPCION 2: Recorriendo por elemento (mas "pythonico")
10 for fila_actual in matriz:
11     for elemento in fila_actual:
12         print(elemento, end="")
13     print()

```

python3 mai...
elemento en (0,0) es 1
elemento en (0,1) es 2
elemento en (0,2) es 3
elemento en (1,0) es 4
elemento en (1,1) es 5
elemento en (1,2) es 6
elemento en (2,0) es 7
elemento en (2,1) es 8
elemento en (2,2) es 9
123
456
789

Código: Recorriendo Con Indices

```

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

```

```

1 teclado= [
2     ["[0]", "[2]", "[3]"],
3     ["[4]", "[5]", "[6]"],
4     ["[7]", "[8]", "[9]"]
5 ]
6 # OPCION 1: Recorriendo con indices
7 num_filas = len(teclado)
8 num_columnas = len(teclado[0])
9 for i in range(num_filas):
10     for j in range(num_columnas):
11         tecla=teclado[i][j]
12         print(f"Tecla en ({i},{j}) es {tecla}")
13 # OPCION 2: Recorriendo por elemento (mas "pythonico")
14 for fila in teclado:
15     for tecla in fila:
16         print(tecla, end="")
17     print()

```

Tecla en (0,0) es [0]
 Tecla en (0,1) es [2]
 Tecla en (0,2) es [3]
 Tecla en (1,0) es [4]
 Tecla en (1,1) es [5]
 Tecla en (1,2) es [6]
 Tecla en (2,0) es [7]
 Tecla en (2,1) es [8]
 Tecla en (2,2) es [9]
 [0][2][3]
 [4][5][6]
 [7][8][9]

Código:Matriz_RecorriendoConIndices

```

matriz= [0, 2, 3], [4, 5, 6], [7, 8, 9]
# OPCION 1: Recorriendo con indices
num_filas = len(matriz)
num_columnas = len(matriz[0])
for i in range(num_filas):
    for j in range(num_columnas):
        elemento = matriz[i][j]
        print(f"elemento en ({i},{j}) es {elemento}")
# OPCION 2: Recorriendo por elemento (mas "pythonico")
for fila_actual in matriz:
    for elemento in fila_actual:
        print(elemento, end="\t")
    print()

```

```

1 matriz= [0, 2, 3], [4, 5, 6], [7, 8, 9]
2 # OPCION 1: Recorriendo con indices
3 num_filas = len(matriz)
4 num_columnas = len(matriz[0])
5 for i in range(num_filas):
6     for j in range(num_columnas):
7         elemento = matriz[i][j]
8         print(f"elemento en ({i},{j}) es {elemento}")
9 # OPCION 2: Recorriendo por elemento (mas "pythonico")
10 for fila_actual in matriz:
11     for elemento in fila_actual:
12         print(elemento, end="\t")
13     print()

```

elemento en (0,0) es 0
 elemento en (0,1) es 2
 elemento en (0,2) es 3
 elemento en (1,0) es 4
 elemento en (1,1) es 5
 elemento en (1,2) es 6
 elemento en (2,0) es 7
 elemento en (2,1) es 8
 elemento en (2,2) es 9
 0 2 3
 4 5 6
 7 8 9

Código:matriz_telefono

Definimos la función que suma los elementos de la diagonal principal de una matriz cuadrada

```
def sumar_diagonal_principal(matriz):
```

```

"""
Esta función recibe una matriz cuadrada (misma cantidad de filas y columnas)
y retorna la suma de los elementos en su diagonal principal.
Ejemplo:
matriz = [[1, 2],
          [3, 4]]
diagonal principal: 1 y 4 → suma = 5
"""

suma = 0
for i in range(len(matriz)):
    suma += matriz[i][i] # Accede al elemento en la posición (i, i)
return suma

# Función de prueba para verificar que sumar_diagonal_principal funciona
correctamente
def probar_suma_diagonal_principal():
    print("\nProbando sumar_diagonal_principal...")
    # Caso 1: matriz 3x3 con números consecutivos
    m1 = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
    assert sumar_diagonal_principal(m1) == 15 # 1 + 5 + 9
    # Caso 2: matriz 2x2 con ceros y valores definidos
    m2 = [[10, 0], [0, 20]]
    assert sumar_diagonal_principal(m2) == 30 # 10 + 20
    # Caso borde: matriz 1x1
    m3 = [[5]]
    assert sumar_diagonal_principal(m3) == 5 # Solo un elemento en la diagonal
    print("¡Pruebas para sumar_diagonal_principal pasaron!")

# Llamamos a la función para ejecutar las pruebas
probar_suma_diagonal_principal()

```

The screenshot shows a Replit Python environment with a file named 'main.py'. The code in the file is the same as the one in the previous block. The console output shows the test results: 'Probando sumar_diagonal_principal... ¡Pruebas para sumar_diagonal_principal pasaron!'.

Codigo: Suma diagonal

```

# Definimos la función que suma los elementos por cada fila de la matriz
def sumar_por_filas(matriz):
    """
    Esta función recibe una matriz (lista de listas)

```


y devuelve una lista con la suma de cada fila.

Ejemplo:

```
matriz = [[1, 2, 3], [4, 5, 6]]
```

```
resultado = [6, 15]
```

```
"""
```

```
resultado = []
```

```
for fila in matriz:
```

```
    suma_fila = sum(fila) # Suma todos los elementos de la fila
```

```
    resultado.append(suma_fila)
```

```
return resultado
```

```
# Función de prueba para verificar que sumar_por_filas funciona correctamente
```

```
def probar_suma_por_filas():
```

```
    print("\nProbando sumar_por_filas...")
```

```
    # Caso 1: matriz con 3 filas y 3 columnas
```

```
    m1 = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

```
    assert sumar_por_filas(m1) == [6, 15, 24] # 1+2+3, 4+5+6, 7+8+9
```

```
    # Caso 2: matriz con pares repetidos
```

```
    m2 = [[10, 10], [20, 20], [30, 30]]
```

```
    assert sumar_por_filas(m2) == [20, 40, 60]
```

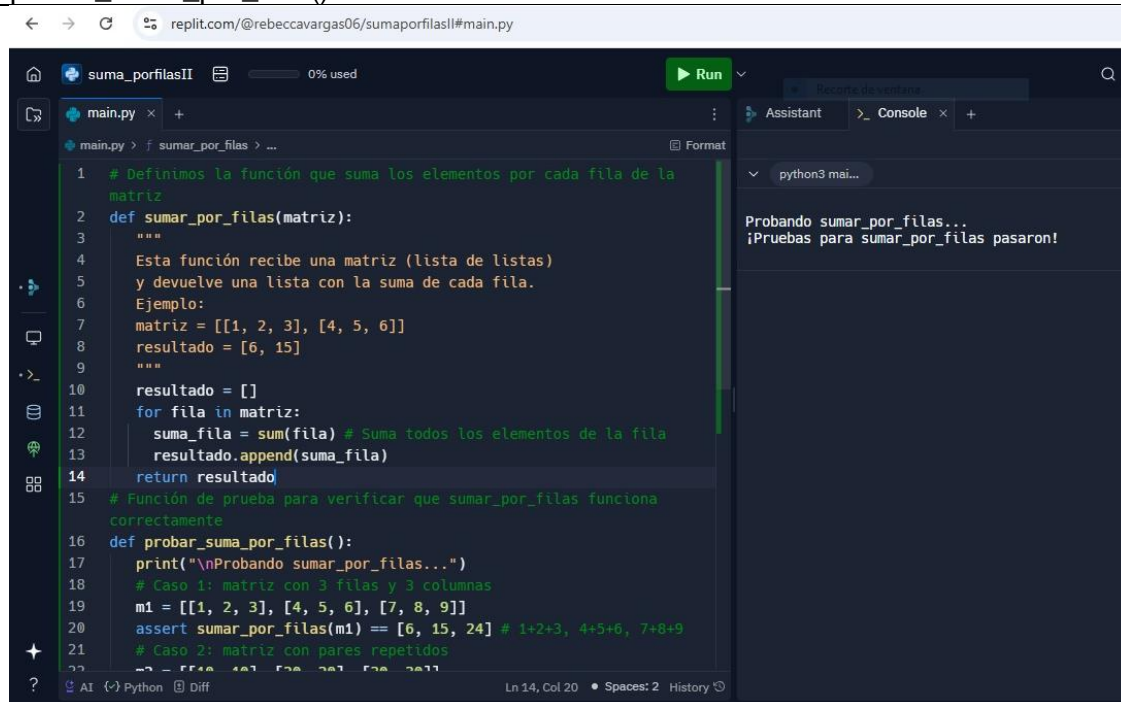
```
    # Caso borde: matriz vacía
```

```
    assert sumar_por_filas([]) == [] # No hay filas que sumar
```

```
    print("¡Pruebas para sumar_por_filas pasaron!")
```

```
# Llamamos a la función para ejecutar las pruebas
```

```
probar_suma_por_filas()
```



```
replit.com/@rebeccavargas06/sumaporfilasII#main.py
```

```
suma_porfilasII 0% used
```

```
main.py x +
```

```
main.py > f suma_por_filas > ...
```

```
1  # Definimos la función que suma los elementos por cada fila de la
2  matriz
3  def sumar_por_filas(matriz):
4  """
5  Esta función recibe una matriz (lista de listas)
6  y devuelve una lista con la suma de cada fila.
7  Ejemplo:
8  matriz = [[1, 2, 3], [4, 5, 6]]
9  resultado = [6, 15]
10 """
11 resultado = []
12 for fila in matriz:
13     suma_fila = sum(fila) # Suma todos los elementos de la fila
14     resultado.append(suma_fila)
15 return resultado
16
17 # Función de prueba para verificar que sumar_por_filas funciona
18 correctamente
19 def probar_suma_por_filas():
20     print("\nProbando sumar_por_filas...")
21     # Caso 1: matriz con 3 filas y 3 columnas
22     m1 = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
23     assert sumar_por_filas(m1) == [6, 15, 24] # 1+2+3, 4+5+6, 7+8+9
24     # Caso 2: matriz con pares repetidos
25     m2 = [[10, 10], [20, 20], [30, 30]]
26     assert sumar_por_filas(m2) == [20, 40, 60]
27     # Caso borde: matriz vacía
28     assert sumar_por_filas([]) == [] # No hay filas que sumar
29     print("¡Pruebas para sumar_por_filas pasaron!")
30
31 # Llamamos a la función para ejecutar las pruebas
32 probar_suma_por_filas()
```

```
python3 mai...
Probando sumar_por_filas...
¡Pruebas para sumar_por_filas pasaron!
```

Codigo: Suma por filas

```
# Definimos la función que suma todos los elementos de una matriz
```

```
def sumar_total_matriz(matriz):
```

```
    """
```

```
    Esta función recibe una matriz (lista de listas)
```

```
    y retorna la suma total de todos sus elementos.
```

```
    Ejemplo:
```

```
    matriz = [[1, 2], [3, 4]]
```

```

resultado = 10
"""

total = 0
for fila in matriz:
    for elemento in fila:
        total += elemento
return total
# Función para probar que sumar_total_matriz funciona correctamente
def probar_suma_total():
    print("Probando sumar_total_matriz...")
    # Caso 1: matriz normal
    m1 = [[1, 2, 3], [4, 5, 6]]
    assert sumar_total_matriz(m1) == 21 # 1+2+3+4+5+6 = 21
    # Caso 2: matriz con negativos y ceros
    m2 = [[-1, 0, 1], [10, -5, 5]]
    assert sumar_total_matriz(m2) == 10 # -1+0+1+10-5+5 = 10
    # Casos borde o límites
    assert sumar_total_matriz([]) == 0 # Matriz con una fila vacía
    assert sumar_total_matriz([]) == 0 # Matriz completamente vacía
    assert sumar_total_matriz([[42]]) == 42 # Matriz de un solo elemento
    print("¡Pruebas para sumar_total_matriz pasaron! ")
# Llamamos a la función de pruebas
probar_suma_total()

```

The screenshot shows a Replit Python environment with a file named `main.py`. The code in the editor is the same as the one in the first block. The console output on the right shows the execution results: "Probando sumar_total_matriz..." and "¡Pruebas para sumar_total_matriz pasaron!". The status bar at the bottom indicates "Ln 5, Col 51 • Spaces: 2 History".

Codigo: Analisis Matricial

```

def transponer_matriz(matriz):
    if not matriz or not matriz[0]:
        return []
    num_filas = len(matriz)
    num_columnas = len(matriz[0])

    matriz_transpuesta = []
    for j in range(num_columnas):

```



```
nueva_fila = []
for i in range(num_filas):
    nueva_fila.append(matriz[i][j])
matriz_transpuesta.append(nueva_fila)
return matriz_transpuesta
```

```
m1 = [[1, 2, 3], [4, 5, 6]]
t1 = transponer_matriz(m1)
assert t1 == [[1, 4], [2, 5], [3, 6]]
print("Prueba 1 (2x3) pasada!")
```

The screenshot shows a code editor with a file named 'main.py'. The code defines a function 'transponer_matriz' that takes a matrix and returns its transpose. It uses nested loops to iterate over the rows and columns. Below the function, a test is performed using a 2x3 matrix 'm1' and its transpose 't1'. The console output shows 'Prueba 1 (2x3) pasada!'.

```
def transponer_matriz(matriz):
    if not matriz or not matriz[0]:
        return []
    num_filas = len(matriz)
    num_columnas = len(matriz[0])

    matriz_transpuesta = []
    for j in range(num_columnas):
        nueva_fila = []
        for i in range(num_filas):
            nueva_fila.append(matriz[i][j])
        matriz_transpuesta.append(nueva_fila)
    return matriz_transpuesta

m1 = [[1, 2, 3], [4, 5, 6]]
t1 = transponer_matriz(m1)
assert t1 == [[1, 4], [2, 5], [3, 6]]
print("Prueba 1 (2x3) pasada!")
```

Código: tansponer una matriz

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

The screenshot shows a code editor with a file named 'main.py'. The code creates a dictionary 'producto' and iterates over its keys and values. The console output shows the keys and then the key-value pairs with the key capitalized.

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

Código: Diccionario bucle for

```

#Paso 1: Variables Globales
lista_de_tareas = []
proximo_id_tarea = 1 # Para generar IDs únicos
#Paso 2: Implementar agregar_tarea
def agregar_tarea(descripcion, prioridad="media"):
    global proximo_id_tarea # ¡Necesario para modificar una variable global!
    nueva_tarea = {
        "id": proximo_id_tarea,
        "descripcion": descripcion,
        "completada": False,
        "prioridad": prioridad
    }
    lista_de_tareas.append(nueva_tarea)
    proximo_id_tarea += 1
    print(f" Tarea '{descripcion}' añadida con éxito.")
#Paso 3: Implementar mostrar_tareas
def mostrar_tareas():
    print("\n--- LISTA DE TAREAS ---")
    if not lista_de_tareas:
        print("¡No hay tareas pendientes! ¡A disfrutar!")
        return

    for tarea in lista_de_tareas:
        estado = " " if tarea["completada"] else " "
        print(f"{estado} ID: {tarea['id']} | {tarea['descripcion']} (Prioridad: {tarea['prioridad']})")
        print("-----")
#Prueba tus funciones:
agregar_tarea("Estudiar para el examen de Cálculo")
agregar_tarea("Hacer las compras", prioridad="alta")
mostrar_tareas()

```

The screenshot shows a Replit Python environment with the following code in `main.py`:

```

1 #Paso 1: Variables Globales
2 lista_de_tareas = []
3 proximo_id_tarea = 1 # Para generar IDs únicos
4 #Paso 2: Implementar agregar_tarea
5 def agregar_tarea(descripcion, prioridad="media"):
6     global proximo_id_tarea # ¡Necesario para modificar una variable global!
7     nueva_tarea = {
8         "id": proximo_id_tarea,
9         "descripcion": descripcion,
10        "completada": False,
11        "prioridad": prioridad
12    }
13    lista_de_tareas.append(nueva_tarea)
14    proximo_id_tarea += 1
15    print(f" Tarea '{descripcion}' añadida con éxito.")
16 #Paso 3: Implementar mostrar_tareas
17 def mostrar_tareas():
18     print("\n--- LISTA DE TAREAS ---")
19     if not lista_de_tareas:
20         print("¡No hay tareas pendientes! ¡A disfrutar!")
21         return
22     for tarea in lista_de_tareas:
23         estado = " " if tarea["completada"] else " "
24         print(f"{estado} ID: {tarea['id']} | {tarea['descripcion']} (Prioridad: {tarea['prioridad']})")
25         print("-----")
26 #Prueba tus funciones:
27 agregar_tarea("Estudiar para el examen de Cálculo")
28 agregar_tarea("Hacer las compras", prioridad="alta")
29 mostrar_tareas()

```

The console output shows the following results:

```

Tarea 'Estudiar para el examen de Cálculo' añadida con éxito.
Tarea 'Hacer las compras' añadida con éxito.

--- LISTA DE TAREAS ---
ID: 1 | Estudiar para el examen de Cálculo (Prioridad: media)
ID: 2 | Hacer las compras (Prioridad: alta)
-----

```

Codigo: Buscar Tarea por ID

```

#Paso 1: Variables Globales
lista_de_tareas = []
proximo_id_tarea = 1 # Para generar IDs únicos
#Paso 2: Implementar agregar_tarea
def agregar_tarea(descripcion, prioridad="media"):
    global proximo_id_tarea # ¡Necesario para modificar una variable global!
    nueva_tarea = {
        "id": proximo_id_tarea,
        "descripcion": descripcion,
        "completada": False,
        "prioridad": prioridad
    }
    lista_de_tareas.append(nueva_tarea)
    proximo_id_tarea += 1
    print(f" Tarea '{descripcion}' añadida con éxito.")
#Paso 3: Implementar mostrar_tareas
def mostrar_tareas():
    print("\n--- LISTA DE TAREAS ---")
    if not lista_de_tareas:
        print("¡No hay tareas pendientes! ¡A disfrutar!")
        return

    for tarea in lista_de_tareas:
        estado = " " if tarea["completada"] else " "
        print(f"{estado} ID: {tarea['id']} | {tarea['descripcion']} (Prioridad: {tarea['prioridad']})")
        print("-----")
#Prueba tus funciones:
agregar_tarea("Estudiar para el examen de Cálculo")
agregar_tarea("Hacer las compras", prioridad="alta")
mostrar_tareas()
#Paso 4: Implementar buscar_tarea_por_id
def buscar_tarea_por_id(id_buscado):
    """Recorre la lista de tareas y devuelve el diccionario de la tarea
    que coincide con el id_buscado. Si no la encuentra, devuelve None."""
    for tarea in lista_de_tareas:
        if tarea["id"] == id_buscado:
            return tarea # ¡Éxito! Devolvemos el diccionario completo
    return None # Si el bucle termina, no se encontró
#Prueba tu función de búsqueda:
# Asumiendo que ya agregaste tareas con IDs 1 y 2...
tarea_encontrada = buscar_tarea_por_id(1)
if tarea_encontrada:
    print(f"\nBúsqueda exitosa: {tarea_encontrada['descripcion']}")
else:
    print("\nBúsqueda fallida: Tarea no encontrada.")
tarea_fantasma = buscar_tarea_por_id(99)
if not tarea_fantasma:
    print("Búsqueda de tarea inexistente funcionó correctamente.")
#Paso 5: Implementar marcar_tarea_completada
def marcar_tarea_completada(id_tarea):
    # ¡Reutilizamos nuestra función de búsqueda!
    tarea = buscar_tarea_por_id(id_tarea)
    if tarea: # Si la búsqueda devolvió un diccionario (no None)
        tarea["completada"] = True

```

```

        print(f" Tarea '{tarea['descripcion']}' marcada como completada.")
    else:
        print(f" Error: No se encontró la tarea con ID {id_tarea}.")
#Paso 6: Implementar eliminar_tarea
def eliminar_tarea(id_tarea):
    tarea = buscar_tarea_por_id(id_tarea)
    if tarea:
        lista_de_tareas.remove(tarea)
        print(f" Tarea '{tarea['descripcion']}' eliminada.")
    else:
        print(f" Error: No se encontró la tarea con ID {id_tarea}.")
#Prueba todo el flujo:
mostrar_tareas()
marcar_tarea_completada(1)
mostrar_tareas() # Debería mostrar la tarea 1 como completada
eliminar_tarea(2)
mostrar_tareas() # La tarea 2 ya no debería aparecer
marcar_tarea_completada(99) # Probar con un ID que no existe

```

```

1 #Paso 1: Variables Globales
2 lista_de_tareas = []
3 proximo_id_tarea = 1 # Para generar IDs únicos
4 #Paso 2: Implementar agregar_tarea
5 def agregar_tarea(descripcion, prioridad="media"):
6     global proximo_id_tarea # Necesario para modificar una variable global!
7     nueva_tarea = {
8         "id": proximo_id_tarea,
9         "descripcion": descripcion,
10        "completada": False,
11        "prioridad": prioridad
12    }
13    lista_de_tareas.append(nueva_tarea)
14    proximo_id_tarea += 1
15    print(f" Tarea '{descripcion}' añadida con éxito.")
16 #Paso 3: Implementar mostrar_tareas
17 def mostrar_tareas():
18     print("\n--- LISTA DE TAREAS ---")
19     if not lista_de_tareas:
20         print("¡No hay tareas pendientes! ¡A disfrutar!")
21     return
22
23 # Pruebas
24 agregar_tarea("Estudiar para el examen de Cálculo")
25 agregar_tarea("Hacer las compras", prioridad="alta")
26 mostrar_tareas()
27 marcar_tarea_completada(1)
28 mostrar_tareas()
29 eliminar_tarea(2)
30 mostrar_tareas()
31 marcar_tarea_completada(99)

```

Console Output:

```

Tarea 'Estudiar para el examen de Cálculo' añadida con éxito.
Tarea 'Hacer las compras' añadida con éxito.
--- LISTA DE TAREAS ---
ID: 1 | Estudiar para el examen de Cálculo (Prioridad: media)
ID: 2 | Hacer las compras (Prioridad: alta)
Búsqueda exitosa: Estudiar para el examen de Cálculo
Búsqueda de tarea inexistente funcionó correctamente.
--- LISTA DE TAREAS ---
ID: 1 | Estudiar para el examen de Cálculo (Prioridad: media)
ID: 2 | Hacer las compras (Prioridad: alta)
Tarea 'Estudiar para el examen de Cálculo' marcada como completada.
--- LISTA DE TAREAS ---
ID: 1 | Estudiar para el examen de Cálculo (Prioridad: media)
ID: 2 | Hacer las compras (Prioridad: alta)
Tarea 'Hacer las compras' eliminada.
--- LISTA DE TAREAS ---
ID: 1 | Estudiar para el examen de Cálculo (Prioridad: media)
Error: No se encontró la tarea con ID 99.

```

código: buscar tareas por ID

class Libro:

"""

Clase que representa un libro con sus atributos principales.

"""

def __init__(self, titulo, autor, isbn, paginas):

"""

Constructor de la clase Libro.

Args:

titulo (str): Título del libro

autor (str): Autor del libro

isbn (str): ISBN del libro

paginas (int): Número de páginas del libro

"""

```

# Crear los atributos de instancia correspondientes
self.titulo = titulo
self.autor = autor
self.isbn = isbn
self.paginas = paginas

# Atributo extra que se inicializa siempre por defecto
self.disponible = True

def mostrar_info(self):
    """
    Método que imprime todos los atributos del libro de forma clara y formateada.
    """

    print("=" * 50)
    print("INFORMACIÓN DEL LIBRO")
    print("=" * 50)
    print(f"Título: {self.titulo}")
    print(f"Autor: {self.autor}")
    print(f"ISBN: {self.isbn}")
    print(f"Páginas: {self.paginas}")
    print(f"Disponible: {'Sí' if self.disponible else 'No'}")
    print("=" * 50)

# Ejemplo de uso (no te preocupes por crear objetos todavía, ¡solo define la clase!)
if __name__ == "__main__":
    # Creación de ejemplo para demostrar el funcionamiento
    libro1 = Libro("Cien años de soledad", "Gabriel García Márquez", "978-0307474728", 417)
    libro1.mostrar_info()

    # Cambiar disponibilidad
    libro1.disponible = False
    print("\nDespués de cambiar disponibilidad:")
    libro1.mostrar_info()

```

The screenshot shows a Replit IDE with a Python script and its output. The script defines a `Libro` class with attributes `titulo`, `autor`, `isbn`, `paginas`, and `disponible`. It creates an instance `libro1` with the title "Cien años de soledad" by Gabriel García Márquez, ISBN 978-0307474728, and 417 pages. The output shows the book's details and its availability status before and after being set to False.

```

class Libro:
    """
    Clase que representa un libro con sus atributos principales.
    """

    def __init__(self, titulo, autor, isbn, paginas):
        """
        Constructor de la clase Libro.
        """
        self.titulo = titulo
        self.autor = autor
        self.isbn = isbn
        self.paginas = paginas

        # Atributo extra que se inicializa siempre por defecto
        self.disponible = True

# Ejemplo de uso (no te preocupes por crear objetos todavía, ¡solo define la clase!)
if __name__ == "__main__":
    # Creación de ejemplo para demostrar el funcionamiento
    libro1 = Libro("Cien años de soledad", "Gabriel García Márquez", "978-0307474728", 417)
    libro1.mostrar_info()

    # Cambiar disponibilidad
    libro1.disponible = False
    print("\nDespués de cambiar disponibilidad:")
    libro1.mostrar_info()

```

Output:

```

=====
INFORMACIÓN DEL LIBRO
=====
Título: Cien años de soledad
Autor: Gabriel García Márquez
ISBN: 978-0307474728
Páginas: 417
Disponible: Sí
=====

Después de cambiar disponibilidad:

=====
INFORMACIÓN DEL LIBRO
=====
Título: Cien años de soledad
Autor: Gabriel García Márquez
ISBN: 978-0307474728
Páginas: 417
Disponible: No
=====

```

código: poo_ej1

Ejercicio 2: Creación de instancias y métodos de comportamiento

class Libro:

"""

Clase que representa un libro con sus atributos principales.

"""

def __init__(self, titulo, autor, isbn, paginas):

"""

Constructor de la clase Libro.

"""

self.titulo = titulo

self.autor = autor

self.isbn = isbn

self.paginas = paginas

self.disponible = True

def mostrar_info(self):

"""

Método que imprime todos los atributos del libro de forma clara y formateada.

"""

print("=" * 50)

print("INFORMACIÓN DEL LIBRO")

print("=" * 50)

print(f"Título: {self.titulo}")

print(f"Autor: {self.autor}")

print(f"ISBN: {self.isbn}")

print(f"Páginas: {self.paginas}")

print(f"Disponible: {'Sí' if self.disponible else 'No'}")

print("=" * 50)

def prestar_libro(self):

"""

Método para prestar el libro. Cambia disponible a False si está disponible.

"""

if self.disponible == True:

self.disponible = False

print(f"El libro '{self.titulo}' ha sido prestado.")

else:

print(f"El libro '{self.titulo}' ya está prestado.")

def devolver_libro(self):

"""

Método para devolver el libro. Cambia disponible a True si estaba prestado.

"""

if self.disponible == False:

self.disponible = True

print(f"El libro '{self.titulo}' ha sido devuelto.")

else:

print(f"El libro '{self.titulo}' ya estaba disponible.")

1. Definición de clase Libro (arriba) ✓

2. Crear dos objetos Libro diferentes


```
libro1 = Libro("El Principito", "Antoine de Saint-Exupéry", "978-3-14-046401-7", 120)
libro2 = Libro("Raza de Bronce", "Alcides Arguedas", "978-99905-2-213-9", 250)

# 3. Acceder y mostrar algunos de sus atributos directamente
print(f"\nEl autor del primer libro es: {libro1.autor}")
print(f"El ISBN del segundo libro es: {libro2.isbn}")

# 4. Llamar al método mostrar_info() para cada uno de los objetos
print("\n--- Mostrando información completa ---")
libro1.mostrar_info()
libro2.mostrar_info()

# 5. Añadir métodos de comportamiento (ya implementados arriba) ✓

# 6. Prueba los nuevos métodos con tus objetos libro1 y libro2
print("\n--- Probando métodos de comportamiento ---")

# Prestar libro1
libro1.prestar_libro()

# Intentar prestar libro1 otra vez (ya prestado)
libro1.prestar_libro()

# Devolver libro1
libro1.devolver_libro()

# Intentar devolver libro1 otra vez (ya disponible)
libro1.devolver_libro()

print("\n--- Probando con libro2 ---")

# Prestar libro2
libro2.prestar_libro()

# Devolver libro2
libro2.devolver_libro()

print("\n--- Estado final de los libros ---")
libro1.mostrar_info()
libro2.mostrar_info()
```

The screenshot shows a Replit IDE with a Python file named `main.py`. The code defines a class `Libro` with attributes `titulo`, `autor`, `isbn`, `paginas`, and `disponible`. It includes an `__init__` method for initialization and a `mostrar_info` method for displaying book information. The console output shows the execution of the code, demonstrating the creation of two book objects, `libro1` and `libro2`, and their respective information display.

```
1 # Ejercicio 2: Creación de instancias y métodos de comportamiento
2
3 class Libro:
4     """
5     Clase que representa un libro con sus atributos principales.
6     """
7
8     def __init__(self, titulo, autor, isbn, paginas):
9         """
10         Constructor de la clase Libro.
11         """
12         self.titulo = titulo
13         self.autor = autor
14         self.isbn = isbn
15         self.paginas = paginas
16         self.disponible = True
17
18     def mostrar_info(self):
19         """
20         Método que imprime todos los atributos del libro de forma
21         clara y formateada.
22         """
23         print("=" * 50)
24         print("INFORMACIÓN DEL LIBRO")
25         print("=" * 50)
26         print(f"Título: {self.titulo}")
27         print(f"Autor: {self.autor}")
28         print(f"ISBN: {self.isbn}")
29         print(f"Páginas: {self.paginas}")
30         print(f"Disponible: {'Sí' if self.disponible else 'No'}")
31         print("=" * 50)
```

Console Output:

```
--- Probando métodos de comportamiento ---
El libro 'El Principito' ha sido prestado.
El libro 'El Principito' ya está prestado.
El libro 'El Principito' ha sido devuelto.
El libro 'El Principito' ya estaba disponible.

--- Probando con libro2 ---
El libro 'Raza de Bronce' ha sido prestado.
El libro 'Raza de Bronce' ha sido devuelto.

--- Estado final de los libros ---

INFORMACIÓN DEL LIBRO
=====
Título: El Principito
Autor: Antoine de Saint-Exupéry
ISBN: 978-3-14-046401-7
Páginas: 120
Disponible: Sí

INFORMACIÓN DEL LIBRO
=====
Título: Raza de Bronce
Autor: Alcides Arguedas
ISBN: 978-99905-2-213-9
Páginas: 250
Disponible: Sí
```

código: poo_ej2

class Libro:

"""

Clase que representa un libro con sus atributos principales.

"""

def __init__(self, titulo, autor, isbn, paginas):

"""

Constructor de la clase Libro.

Args:

titulo (str): Título del libro

autor (str): Autor del libro

isbn (str): ISBN del libro

paginas (int): Número de páginas del libro

"""

self.titulo = titulo

self.autor = autor

self.isbn = isbn

self.paginas = paginas

self.disponible = True

def mostrar_info(self):

"""

Método que imprime todos los atributos del libro de forma clara y formateada.

"""

print("=" * 50)

print("INFORMACIÓN DEL LIBRO")

print("=" * 50)

print(f"Título: {self.titulo}")

print(f"Autor: {self.autor}")

print(f"ISBN: {self.isbn}")

print(f"Páginas: {self.paginas}")

print(f"Disponible: {'Sí' if self.disponible else 'No'}")

print("=" * 50)

```

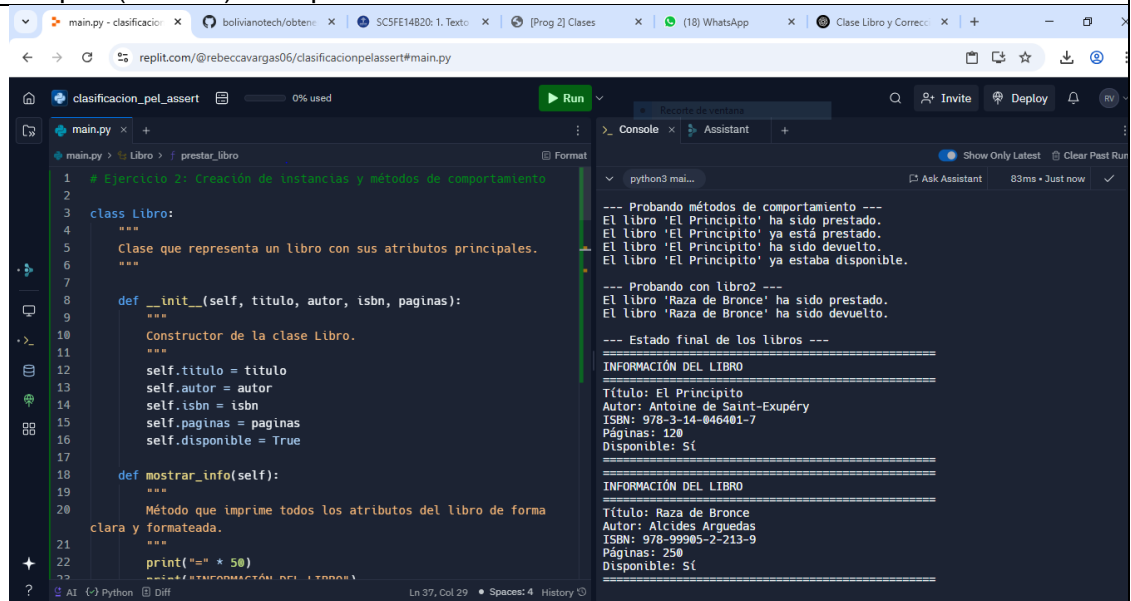
if __name__ == "__main__":
    # Crear objetos de tipo Libro
    libro1 = Libro("Cien años de soledad", "Gabriel García Márquez", "978-0307474728", 417)
    libro2 = Libro("1984", "George Orwell", "978-0451524935", 328)
    libro3 = Libro("El Principito", "Antoine de Saint-Exupéry", "978-0156013987", 96)

    # Crear una lista vacía
    mi_biblioteca = []

    # Añadir libros a la lista
    mi_biblioteca.append(libro1)
    mi_biblioteca.append(libro2)
    mi_biblioteca.append(libro3)

    # Mostrar el inventario completo
    print("\n\n--- INVENTARIO COMPLETO DE LA BIBLIOTECA ---")
    for libro_actual in mi_biblioteca:
        libro_actual.mostrar_info()
        print("=" * 20) # Separador

```



The screenshot shows a Replit Python environment with the following code in `main.py`:

```

1 # Ejercicio 2: Creación de instancias y métodos de comportamiento
2
3 class Libro:
4     """
5     Clase que representa un libro con sus atributos principales.
6     """
7
8     def __init__(self, titulo, autor, isbn, paginas):
9
10        Constructor de la clase Libro.
11        """
12        self.titulo = titulo
13        self.autor = autor
14        self.isbn = isbn
15        self.paginas = paginas
16        self.disponible = True
17
18    def mostrar_info(self):
19        """
20        Método que imprime todos los atributos del libro de forma
21        clara y formateada.
22        """
23        print("=" * 50)
24        print(f"INFORMACIÓN DEL LIBRO")
25        print(f"Título: {self.titulo}")
26        print(f"Autor: {self.autor}")
27        print(f"ISBN: {self.isbn}")
28        print(f"Páginas: {self.paginas}")
29        print(f"Disponible: {'Sí' if self.disponible else 'No'}")
30        print("=" * 50)

```

The console output shows the following results:

```

--- Probando métodos de comportamiento ---
El libro 'El Principito' ha sido prestado.
El libro 'El Principito' ya está prestado.
El libro 'El Principito' ha sido devuelto.
El libro 'El Principito' ya estaba disponible.

--- Probando con libro2 ---
El libro 'Raza de Bronce' ha sido prestado.
El libro 'Raza de Bronce' ha sido devuelto.

--- Estado final de los libros ---

INFORMACIÓN DEL LIBRO
=====
Título: El Principito
Autor: Antoine de Saint-Exupéry
ISBN: 978-0-14-046401-7
Páginas: 120
Disponible: Sí
=====

INFORMACIÓN DEL LIBRO
=====
Título: Raza de Bronce
Autor: Alcides Arguedas
ISBN: 978-9905-2-213-9
Páginas: 250
Disponible: Sí
=====

```

código: po_ej3

```

#Paso 1: Variables Globales
lista_de_tareas = []
proximo_id_tarea = 1 # Para generar IDs únicos
#Paso 2: Implementar agregar_tarea
def agregar_tarea(descripcion, prioridad="media"):
    global proximo_id_tarea # ¡Necesario para modificar una variable global!
    nueva_tarea = {
        "id": proximo_id_tarea,
        "descripcion": descripcion,
        "completada": False,

```

```

    "prioridad": prioridad
}
lista_de_tareas.append(nueva_tarea)
proximo_id_tarea += 1
print(f" Tarea '{descripcion}' añadida con éxito.")
#Paso 3: Implementar mostrar_tareas
def mostrar_tareas():
    print("\n--- LISTA DE TAREAS ---")
    if not lista_de_tareas:
        print("¡No hay tareas pendientes! ¡A disfrutar!")
        return

    for tarea in lista_de_tareas:
        estado = " " if tarea["completada"] else " "
        print(f"{estado} ID: {tarea['id']} | {tarea['descripcion']} (Prioridad: {tarea['prioridad']})")
        print("-----")
#Prueba tus funciones:
agregar_tarea("Estudiar para el examen de Cálculo")
agregar_tarea("Hacer las compras", prioridad="alta")
mostrar_tareas()
#Paso 4: Implementar buscar_tarea_por_id
def buscar_tarea_por_id(id_buscado):
    """Recorre la lista de tareas y devuelve el diccionario de la tarea
    que coincide con el id_buscado. Si no la encuentra, devuelve None."""
    for tarea in lista_de_tareas:
        if tarea["id"] == id_buscado:
            return tarea # ¡Éxito! Devolvemos el diccionario completo
    return None # Si el bucle termina, no se encontró
#Prueba tu función de búsqueda:
# Asumiendo que ya agregaste tareas con IDs 1 y 2...
tarea_encontrada = buscar_tarea_por_id(1)
if tarea_encontrada:
    print(f"\nBúsqueda exitosa: {tarea_encontrada['descripcion']}")
else:
    print("\nBúsqueda fallida: Tarea no encontrada.")
tarea_fantasma = buscar_tarea_por_id(99)
if not tarea_fantasma:
    print("Búsqueda de tarea inexistente funcionó correctamente.")
#Paso 5: Implementar marcar_tarea_completada
def marcar_tarea_completada(id_tarea):
    # ¡Reutilizamos nuestra función de búsqueda!
    tarea = buscar_tarea_por_id(id_tarea)
    if tarea: # Si la búsqueda devolvió un diccionario (no None)
        tarea["completada"] = True
        print(f" Tarea '{tarea['descripcion']}' marcada como completada.")
    else:
        print(f" Error: No se encontró la tarea con ID {id_tarea}.")
#Paso 6: Implementar eliminar_tarea
def eliminar_tarea(id_tarea):
    tarea = buscar_tarea_por_id(id_tarea)
    if tarea:
        lista_de_tareas.remove(tarea)
        print(f" Tarea '{tarea['descripcion']}' eliminada.")
    else:

```

```

    print(f" Error: No se encontró la tarea con ID {id_tarea}.")
#Prueba todo el flujo:
mostrar_tareas()
marcar_tarea_completada(1)
mostrar_tareas() # Debería mostrar la tarea 1 como completada
eliminar_tarea(2)
mostrar_tareas() # La tarea 2 ya no debería aparecer
marcar_tarea_completada(99) # Probar con un ID que no existe
# ... (definiciones de funciones y pruebas aquí arriba) ...
# ¡Puedes comentar o eliminar las pruebas para tener un programa limpio!
while True:
    print("\n===== MENÚ TO-DO LIST =====")
    print("1. Agregar nueva tarea")
    print("2. Mostrar todas las tareas")
    print("3. Marcar tarea como completada")
    print("4. Eliminar tarea")
    print("0. Salir")
    opcion = input("Elige una opción: ")
    if opcion == '1':
        desc = input("Descripción de la nueva tarea: ")
        prio = input("Prioridad (alta, media, baja): ")
        agregar_tarea(desc, prio)
    elif opcion == '2':
        mostrar_tareas()
    elif opcion == '3':
        id_t = int(input("ID de la tarea a completar: "))
        marcar_tarea_completada(id_t)
    elif opcion == '4':
        id_t = int(input("ID de la tarea a eliminar: "))
        eliminar_tarea(id_t)
    elif opcion == '0':
        print("¡Hasta pronto!")
        break # Rompe el bucle while
    else:
        print(" Opción no válida. Inténtalo de nuevo.")

```

The screenshot shows a Replit IDE with a Python file named `main.py` and a console window. The code implements a To-Do List application with a main menu. The console output shows the menu being displayed, a task being added, and the menu being shown again with the task marked as completed.

```

main.py
1 #Paso 1: Variables Globales
2 lista_de_tareas = []
3 proximo_id_tarea = 1 # Para generar IDs únicos
4 #Paso 2: Implementar agregar_tarea
5 def agregar_tarea(descripcion, prioridad="media"):
6     global proximo_id_tarea # Necesario para modificar una variable global
7     nueva_tarea = {
8         "id": proximo_id_tarea,
9         "descripcion": descripcion,
10        "completada": False,
11        "prioridad": prioridad
12    }
13    lista_de_tareas.append(nueva_tarea)
14    proximo_id_tarea += 1
15    print(f" Tarea '{descripcion}' añadida con éxito.")
16 #Paso 3: Implementar mostrar_tareas
17 def mostrar_tareas():
18     print("\n--- LISTA DE TAREAS ---")
19     if not lista_de_tareas:
20         print("¡No hay tareas pendientes! ¡A disfrutar!")
21     return
22
23 # Las tareas in lista de tareas:

```

Console Output:

```

===== MENÚ TO-DO LIST =====
1. Agregar nueva tarea
2. Mostrar todas las tareas
3. Marcar tarea como completada
4. Eliminar tarea
0. Salir
Elige una opción: 2

--- LISTA DE TAREAS ---
ID: 1 | Estudiar para el examen de Cálculo (Prioridad: media)

===== MENÚ TO-DO LIST =====
1. Agregar nueva tarea
2. Mostrar todas las tareas
3. Marcar tarea como completada
4. Eliminar tarea
0. Salir
Elige una opción: 3
ID de la tarea a completar: 1
Tarea 'Estudiar para el examen de Cálculo' marcada como completada.

===== MENÚ TO-DO LIST =====
1. Agregar nueva tarea
2. Mostrar todas las tareas
3. Marcar tarea como completada
4. Eliminar tarea
0. Salir
Elige una opción: 

```

Código: orquestando con un menú principal

```

# =====
# TO-DO LIST EN PYTHON
# =====

# Paso 1: Variables Globales
lista_de_tareas = []
proximo_id_tarea = 1 # Para generar IDs únicos

# Paso 2: Implementar agregar_tarea
def agregar_tarea(descripcion, prioridad='media'):
    global proximo_id_tarea
    nueva_tarea = {
        "id": proximo_id_tarea,
        "descripcion": descripcion,
        "completada": False,
        "prioridad": prioridad
    }
    lista_de_tareas.append(nueva_tarea)
    proximo_id_tarea += 1
    print(f"✔ Tarea '{descripcion}' añadida con éxito.")

# Paso 3: Implementar mostrar_tareas
def mostrar_tareas():
    print("\n--- 📅 LISTA DE TAREAS ---")
    if not lista_de_tareas:
        print("¡No hay tareas pendientes! ¡A disfrutar!")
        return
    for tarea in lista_de_tareas:
        estado = "✔" if tarea["completada"] else "□"
        print(f"{estado} ID: {tarea['id']} | {tarea['descripcion']} (Prioridad: {tarea['prioridad']})")

# Paso 4: Implementar buscar_tarea_por_id
def buscar_tarea_por_id(id_buscado):
    for tarea in lista_de_tareas:
        if tarea["id"] == id_buscado:
            return tarea
    return None

# Paso 5: Implementar marcar_tarea_completada
def marcar_tarea_completada(id_tarea):
    tarea = buscar_tarea_por_id(id_tarea)
    if tarea:
        tarea["completada"] = True
        print(f"✔ Tarea '{tarea['descripcion']}' marcada como completada.")
    else:
        print(f"✗ Error: No se encontró la tarea con ID {id_tarea}.")

# Paso 6: Implementar eliminar_tarea
def eliminar_tarea(id_tarea):
    tarea = buscar_tarea_por_id(id_tarea)
    if tarea:
        lista_de_tareas.remove(tarea)
        print(f"✔ Tarea '{tarea['descripcion']}' eliminada.")

```



```
else:
    print(f"❌ Error: No se encontró la tarea con ID {id_tarea}.")
```

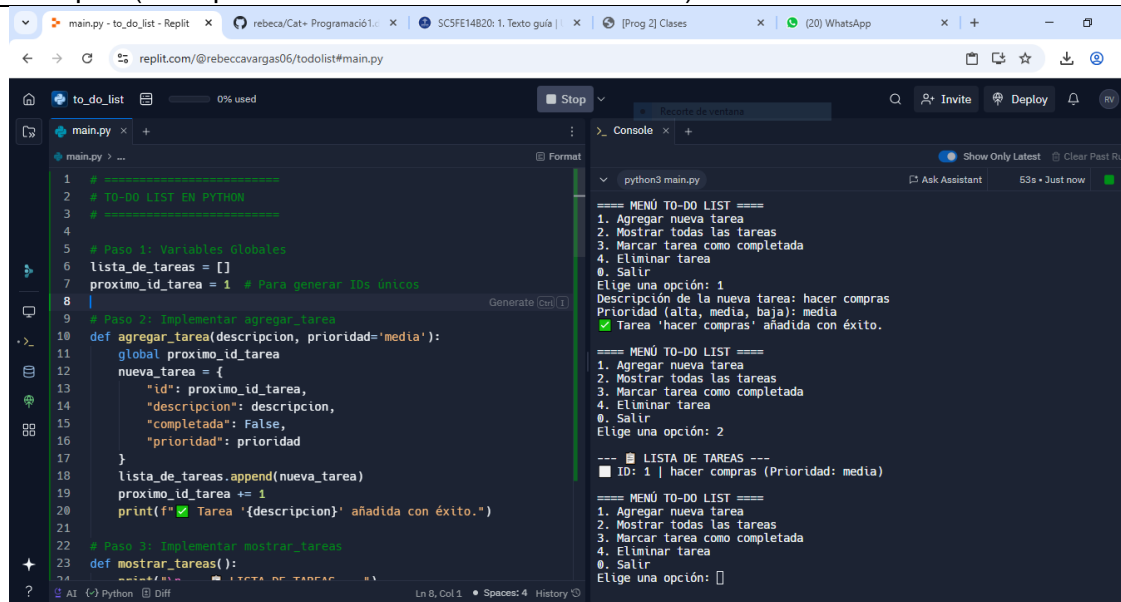
Paso 7: El Bucle Principal del Programa

```
while True:
```

```
    print("\n==== MENÚ TO-DO LIST =====")
    print("1. Agregar nueva tarea")
    print("2. Mostrar todas las tareas")
    print("3. Marcar tarea como completada")
    print("4. Eliminar tarea")
    print("0. Salir")
    opcion = input("Elige una opción: ")
```

```
    if opcion == '1':
        desc = input("Descripción de la nueva tarea: ")
        prio = input("Prioridad (alta, media, baja): ")
        agregar_tarea(desc, prio)
    elif opcion == '2':
        mostrar_tareas()
    elif opcion == '3':
        id_t = int(input("ID de la tarea a completar: "))
        marcar_tarea_completada(id_t)
    elif opcion == '4':
        id_t = int(input("ID de la tarea a eliminar: "))
        eliminar_tarea(id_t)
    elif opcion == '0':
        print("¡Hasta pronto!")
        break
```

```
    else:
        print(f"❌ Opción no válida. Inténtalo de nuevo.")
```



The screenshot shows a Replit IDE with a Python file named `main.py` and a console window. The code in `main.py` includes comments for each step and functions for adding, showing, and deleting tasks. The console output shows the menu being displayed, the user selecting option 1, and a task 'hacer compras' being added successfully. The menu is then displayed again with the new task listed.

```
1 # TO-DO LIST EN PYTHON
2 # =====
3 # Paso 1: Variables Globales
4 lista_de_tareas = []
5 proximo_id_tarea = 1 # Para generar IDs únicos
6
7 # Paso 2: Implementar agregar_tarea
8 def agregar_tarea(description, prioridad='media'):
9     global proximo_id_tarea
10     nueva_tarea = {
11         "id": proximo_id_tarea,
12         "descripcion": description,
13         "completada": False,
14         "prioridad": prioridad
15     }
16     lista_de_tareas.append(nueva_tarea)
17     proximo_id_tarea += 1
18     print(f"✅ Tarea '{description}' añadida con éxito.")
19
20 # Paso 3: Implementar mostrar_tareas
21 def mostrar_tareas():
22     if not lista_de_tareas:
23         print("No hay tareas en la lista.")
24     else:
25         print("\n--- LISTA DE TAREAS ---")
26         for tarea in lista_de_tareas:
27             print(f"ID: {tarea['id']} | {tarea['descripcion']} (Prioridad: {tarea['prioridad']})")
28
29 # Paso 4: Implementar marcar_tarea_completada
30 def marcar_tarea_completada(id_t):
31     for tarea in lista_de_tareas:
32         if tarea['id'] == id_t:
33             tarea['completada'] = True
34             print(f"✅ Tarea '{tarea['descripcion']}' marcada como completada.")
35             return
36     print(f"❌ No se encontró la tarea con ID {id_t}.")
37
38 # Paso 5: Implementar eliminar_tarea
39 def eliminar_tarea(id_t):
40     for tarea in lista_de_tareas:
41         if tarea['id'] == id_t:
42             lista_de_tareas.remove(tarea)
43             print(f"✅ Tarea '{tarea['descripcion']}' eliminada.")
44             return
45     print(f"❌ No se encontró la tarea con ID {id_t}.")
46
47 # Paso 6: Bucle Principal
48 while True:
49     print("\n==== MENÚ TO-DO LIST =====")
50     print("1. Agregar nueva tarea")
51     print("2. Mostrar todas las tareas")
52     print("3. Marcar tarea como completada")
53     print("4. Eliminar tarea")
54     print("0. Salir")
55     opcion = input("Elige una opción: ")
56
57     if opcion == '1':
58         desc = input("Descripción de la nueva tarea: ")
59         prio = input("Prioridad (alta, media, baja): ")
60         agregar_tarea(desc, prio)
61     elif opcion == '2':
62         mostrar_tareas()
63     elif opcion == '3':
64         id_t = int(input("ID de la tarea a completar: "))
65         marcar_tarea_completada(id_t)
66     elif opcion == '4':
67         id_t = int(input("ID de la tarea a eliminar: "))
68         eliminar_tarea(id_t)
69     elif opcion == '0':
70         print("¡Hasta pronto!")
71         break
72     else:
73         print(f"❌ Opción no válida. Inténtalo de nuevo.")
```

Console Output:

```
python3 main.py
==== MENÚ TO-DO LIST =====
1. Agregar nueva tarea
2. Mostrar todas las tareas
3. Marcar tarea como completada
4. Eliminar tarea
0. Salir
Elige una opción: 1
Descripción de la nueva tarea: hacer compras
Prioridad (alta, media, baja): media
✅ Tarea 'hacer compras' añadida con éxito.

==== MENÚ TO-DO LIST =====
1. Agregar nueva tarea
2. Mostrar todas las tareas
3. Marcar tarea como completada
4. Eliminar tarea
0. Salir
Elige una opción: 2

--- LISTA DE TAREAS ---
ID: 1 | hacer compras (Prioridad: media)

==== MENÚ TO-DO LIST =====
1. Agregar nueva tarea
2. Mostrar todas las tareas
3. Marcar tarea como completada
4. Eliminar tarea
0. Salir
Elige una opción: 
```

Código: menú To Do List

```

# Ejercicio 1: Crear y escribir en un archivo de texto llamado "mi_diario.txt"

# Paso 1: Definimos el nombre del archivo
nombre_archivo = "mi_diario.txt"

# Paso 2: Usamos 'with open(...)' en modo escritura ('w')
# Esto Crea el archivo si no existe, y lo sobrescribe si ya existe.
with open(nombre_archivo, "w") as diario_file:
    # Paso 3: Escribimos varias líneas en el archivo usando .write()
    diario_file.write("Querido diario,\n")
    diario_file.write("Hoy aprendí sobre archivos en Python.\n")
    diario_file.write("El modo 'w' borra todo antes de escribir. ¡Qué miedo!\n")
    diario_file.write("¡Pero también es muy útil para comenzar desde cero!\n")

# Paso 4: Confirmamos que se ha terminado de escribir
print("✓ Diario creado y primeras entradas guardadas.")

# Ejercicio 2: Leer el contenido del diario después de escribirlo

# Opción A: Leer todo de golpe (menos recomendada porque no separa líneas bien al mostrar)
# La dejamos comentada como referencia didáctica

# with open(nombre_archivo, "r") as diario_file:
#     contenido = diario_file.read()
#     print("\n--- Contenido completo del diario (modo A) ---")
#     print(contenido)

# Opción B: Leer línea por línea (más clara y controlada)
print("\n--- Contenido del diario (línea por línea) ---")
try:
    with open(nombre_archivo, "r") as diario_file:
        for linea in diario_file:
            print(linea.strip()) # .strip() elimina el salto de línea al final
except FileNotFoundError:
    print(f"✗ Error: El archivo '{nombre_archivo}' no existe.")

# Ejercicio 2: Leer el contenido del archivo "mi_diario.txt"
# Probamos dos formas: opción A (leer todo) y opción B (leer línea por línea)

# -----
# ♠ Opción A: Leer todo de golpe
# -----
# Esta opción carga todo el contenido del archivo como un solo string
# Útil si queremos procesar o mostrar todo junto

print("\n--- Contenido completo del diario (modo A - read()) ---")
try:
    with open(nombre_archivo, "r") as diario_file:
        contenido = diario_file.read() # Lee todo el contenido de una sola vez
        print(contenido)
except FileNotFoundError:
    print(f"✗ Error: El archivo '{nombre_archivo}' no existe.")

```

```

# -----
# 💎 Opción B: Leer línea por línea
# -----
# Esta opción permite manejar cada línea por separado
# Ideal para procesar o mostrar contenido ordenado, sin saltos extra

print("\n--- Contenido del diario (modo B - línea por línea) ---")
try:
    with open(nombre_archivo, "r") as diario_file:
        for linea in diario_file:
            print(linea.strip()) # Eliminamos los '\n' del final para una impresión limpia
except FileNotFoundError:
    print(f"❌ Error: El archivo '{nombre_archivo}' no existe.")

# Ejercicio 3: Añadir nuevas entradas al archivo sin borrar lo anterior (modo 'a')

print("\n📝 Añadiendo nuevas entradas al diario...")

# Abrimos el archivo en modo añadir ('a')
with open(nombre_archivo, "a") as diario_file:
    # Escribimos nuevas líneas. También podemos añadir una línea separadora.
    diario_file.write("\n--- Entrada del 20 de Junio de 2025 ---\n")
    diario_file.write("El modo 'a' es genial para no perder datos.\n")
    diario_file.write("Ahora mi diario puede crecer cada día.\n")

# Confirmamos que se añadieron las nuevas entradas
print("✅ Nuevas entradas guardadas.")

# Verificamos que las nuevas entradas se añadieron correctamente
print("\n🔍 Verificando el contenido final del diario...")

try:
    with open(nombre_archivo, "r") as diario_file:
        for linea in diario_file:
            print(linea.strip())
except FileNotFoundError:
    print(f"❌ Error: El archivo '{nombre_archivo}' no existe.")

```

main.py - clasificacion_pel...rebeca/clase15_archpy at m...SCSFE14820: 1. Texto guía | [Prog 2] Clases(18) WhatsApp

replit.com/@rebeccavargas06/clasificacionpelassert#main.py

clasificacion_pel_assert0% usedRun

main.py

main.py > ...Format

```
1 # Ejercicio 1: Crear y escribir en un archivo de texto llamado
2 "mi_diario.txt"
3 # Paso 1: Definimos el nombre del archivo
4 nombre_archivo = "mi_diario.txt"
5
6 # Paso 2: Usamos 'with open(...)' en modo escritura ('w')
7 # Esto Crea el archivo si no existe, y lo sobrescribe si ya existe.
8 with open(nombre_archivo, "w") as diario_file:
9     # Paso 3: Escribimos varias líneas en el archivo usando .write()
10     diario_file.write("Querido diario,\n")
11     diario_file.write("Hoy aprendí sobre archivos en Python.\n")
12     diario_file.write("El modo 'w' borra todo antes de escribir.
13     ¡Qué miedo!\n")
14     diario_file.write("¡Pero también es muy útil para comenzar desde
15     cero!\n")
16
17 # Paso 4: Confirmamos que se ha terminado de escribir
18 print("✅ Diario creado y primeras entradas guardadas.")
19
20 # Ejercicio 2: Leer el contenido del diario después de escribirlo
21
22 # Opción A: Leer todo de golpe (menos recomendada porque no separa
23     líneas como el usuario)
```

Console

Reporte de ventanaAsk Assistant80ms • Just now

```
python3 mai...
Querido diario,
Hoy aprendí sobre archivos en Python.
El modo 'w' borra todo antes de escribir. ¡Qué miedo!
¡Pero también es muy útil para comenzar desde cero!

--- Contenido completo del diario (modo A - read()) ---
Querido diario,
Hoy aprendí sobre archivos en Python.
El modo 'w' borra todo antes de escribir. ¡Qué miedo!
¡Pero también es muy útil para comenzar desde cero!

--- Contenido del diario (modo B - línea por línea) ---
Querido diario,
Hoy aprendí sobre archivos en Python.
El modo 'w' borra todo antes de escribir. ¡Qué miedo!
¡Pero también es muy útil para comenzar desde cero!

■ Añadiendo nuevas entradas al diario...
✅ Nuevas entradas guardadas.

■ Verificando el contenido final del diario...
Querido diario,
Hoy aprendí sobre archivos en Python.
El modo 'w' borra todo antes de escribir. ¡Qué miedo!
¡Pero también es muy útil para comenzar desde cero!

--- Entrada del 20 de Junio de 2025 ---
El modo 'a' es genial para no perder datos.
Ahora mi diario puede crecer cada día.
```

Código: Diario, nuevas entradas

Código:

Código: