

# UNIVERSIDAD PRIVADA DOMINGO SAVIO



## Actividad 02

**DOCENTE:** Jimmy N. Requena Llorentty

**TURNO:** Mañana

**CARRERA:** Ing. En Sistemas

**MATERIA:** Programación II

**ESTUDIANTE:** Franz Almanza Galindo

**Fecha y hora actual:** 2025-06-21 11:35:42

**Santa Cruz - Bolivia**

## Captura #1 Ordenamiento Burbuja

```
main.py x +
main.py > ...
1 #Codigo Ordenamiento Burbuja
2 def ordenamiento_burbuja(lista):
3
4     n = len(lista) # Cantidad de elementos en la lista
5
6     for i in range(n - 1): # Bucle exterior para las pasadas
7         hubo_intercambio = False # Marca si hubo un intercambio en esta pasada
8
9         # Bucle interior para las comparaciones e intercambios
10        for j in range(n - 1 - i): # Cada pasada evita revisar los últimos ya
ordenados
11            if lista[j] > lista[j + 1]:
12                # ¡Intercambio!
13                lista[j], lista[j + 1] = lista[j + 1], lista[j]
14                hubo_intercambio = True
15
16        if not hubo_intercambio: # Si no hubo ningún intercambio, la lista ya está
ordenada
17            break
18
19    return lista # Opcional: también se puede omitir
20
```

python3 mai... Show Only Latest

Antes: [6, 3, 8, 2, 5]  
Después Ordenamiento Burbuja: [2, 3, 5, 6, 8]

--- Ejecutando pruebas con asserts ---  
Caso 1 (Lista desordenada): Exitoso  
Caso 2 (Lista ya ordenada): Exitoso  
Caso 3 (Lista ordenada a la inversa): Exitoso  
Caso 4 (Lista con elementos duplicados): Exitoso  
Caso borde (Lista vacía): Exitoso  
Caso borde (Lista con un solo elemento): Exitoso

Programa realizado por Franz Almanza

## Captura #2 Ordenamiento por Inserción

```
main.py > | ordenamiento_insercion > ...
#Codigo ordenamiento por insercion
def ordenamiento_insercion(lista):
    for i in range(1, len(lista)):
        valor_actual = lista[i] # La "carta" que vamos a insertar
        posicion_actual = i

        # Desplazar elementos mayores hacia la derecha
        while posicion_actual > 0 and lista[posicion_actual - 1] > valor_actual:
            lista[posicion_actual] = lista[posicion_actual - 1]
            posicion_actual -= 1

        # Insertar la "carta" en su hueco correcto
        lista[posicion_actual] = valor_actual

    return lista

if __name__ == "__main__":
    numeros = [6, 3, 8, 2, 5]
    print("Antes:", numeros)
    ordenamiento_insercion(numeros)
    print("Después Ordenamiento Inserción:", numeros)

    print("\n--- Ejecutando pruebas con asserts ---")
```

python3 mai... Show Only Latest

Antes: [6, 3, 8, 2, 5]  
Después Ordenamiento Inserción: [2, 3, 5, 6, 8]

--- Ejecutando pruebas con asserts ---  
Caso 1 (Lista desordenada): SUCCESS  
Caso 2 (Lista ya ordenada): SUCCESS  
Caso 3 (Lista ordenada a la inversa): SUCCESS  
Caso 4 (Lista con duplicados): SUCCESS  
Caso borde (Lista vacía): SUCCESS  
Caso borde (Lista con un solo elemento): SUCCESS

Programa realizado por Franz Almanza

## Captura #3 Merge Sort

```
#Codigo para el merge sort
def merge_sort(lista):
    # Paso Vencer (Condición Base de la Recursividad):
    if len(lista) <= 1:
        return lista

    # Paso 1: DIVIDIR
    medio = len(lista) // 2
    mitad_izquierda = lista[:medio]
    mitad_derecha = lista[medio:]

    # Paso 2: VENCER (Recursión)
    izquierda_ordenada = merge_sort(mitad_izquierda)
    derecha_ordenada = merge_sort(mitad_derecha)

    # Paso 3: COMBINAR
    # Comentamos la línea de impresión de mezcla para no saturar la salida con los
    asserts
    # print(f"Mezclaría {izquierda_ordenada} y {derecha_ordenada}")
    return merge(izquierda_ordenada, derecha_ordenada)
```

python3 mai... Ask Assistant ✓

Lista original: [8, 3, 5, 1]  
Lista ordenada: [1, 3, 5, 8]

--- Ejecutando pruebas automatizadas con asserts ---  
Caso 1 (Lista vacía): succes  
Caso 2 (Lista con un solo elemento): succes  
Caso 3 (Lista con dos elementos): succes  
Caso 4 (Lista con tres elementos desordenados): succes  
Caso 5 (Lista par): succes  
Caso 6 (Lista en orden descendente): succes  
Caso 7 (Lista ya ordenada): succes  
Caso 8 (Lista con elementos repetidos): succes  
Caso 9 (Lista con enteros negativos y positivos): succes  
Caso 10 (Lista con flotantes): succes

Programa realizado por Franz Almanza

## Captura # Matrices

```
main.py x +
main.py > ...
Format
1 #Codigo Matrices
2 # Como crear la matriz de 3x3
3 teclado = [
4     [1, 2, 3],
5     [4, 5, 6],
6     [7, 8, 9]
7 ]
8
9 # Como imprimir la matriz completa
10 print("Matriz original:")
11 for fila in teclado:
12     print(fila)
13
14 # Acceder a elementos específicos
15 print("\nNúmero en el centro:", teclado[1][1]) # 5
16 print("Número en la esquina inferior derecha:", teclado[2][2]) # 9
17
18 # Modificar el número en la esquina superior izquierda (1 por un 0)
19 teclado[0][0] = 0
20
21 # Imprimir la matriz modificada
22 print("\nMatriz modificada:")
23 print(teclado)
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

python3 mai...

Matriz original:  
[1, 2, 3]  
[4, 5, 6]  
[7, 8, 9]

Número en el centro: 5  
Número en la esquina inferior derecha: 9

Matriz modificada:  
[0, 2, 3]  
[4, 5, 6]  
[7, 8, 9]

Matriz como cuadrícula:  
0 8 9  
4 5 6  
1 2 3

Matriz 5x5 llena de ceros (con bucles):  
[0, 0, 0, 0, 0]  
[0, 0, 0, 0, 0]  
[0, 0, 0, 0, 0]  
[0, 0, 0, 0, 0]  
[0, 0, 0, 0, 0]

Programa realizado por Franz Almanza

## Captura # Sumar total matriz

```
main.py > f probar_suma_total > ...
Format
# Como sumar todos los elementos de una matriz
def sumar_total_matriz(matriz):

    # 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]]
    try:
        assert sumar_total_matriz(m1) == 21, "Fallo en Caso 1: Matriz normal"
        print("Caso 1 (Matriz normal): PASSED")
    except AssertionError as e:
        print(f"Error: {e}")
```

Package

python3 mai...

--- Probando sumar\_total\_matriz ---  
Caso 1 (Matriz normal): PASSED  
Caso 2 (Matriz con negativos y ceros): PASSED  
Caso 3 (Matriz con una fila vacía): PASSED  
Caso 4 (Matriz completamente vacía): PASSED  
Caso 5 (Matriz de un solo elemento): PASSED  
Todas las pruebas para sumar\_total\_matriz han finalizado

Programa realizado por Franz Almanza

## Captura # Sumar fila matriz

```
main.py > f probar_suma_por_filas > ...  
#Codigo para sumar por fila en matriz  
# 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("\n Probando sumar_por_filas")  
  
    # Caso 1: matriz con 3 filas y 3 columnas  
    m1 = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

Packager

python3 mai...

Probando sumar\_por\_filas  
Caso 1 (Matriz 3x3): success  
Caso 2 (Matriz con pares repetidos): success  
Caso 3 (Matriz con números negativos): success  
Caso 4 (Matriz con filas de diferente longitud): success  
Caso 5 (Matriz vacía): success  
Caso 6 (Matriz con filas vacías): success  
Todas las pruebas para sumar\_por\_filas han finalizado  
Programa realizado por Franz Almanza

## Captura # Suma diagonal matriz

```
#Codigo para sumar diagonal en matriz  
# Definimos la función que suma los elementos de la diagonal principal de una matriz  
cuadrada  
def sumar_diagonal_principal(matriz):  
    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("\nPrueba de sumar diagonal")  
  
    # Caso 1: matriz 3x3 con números consecutivos  
    m1 = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]  
    try:  
        assert sumar_diagonal_principal(m1) == 15, "Fallo en Caso 1: Matriz 3x3 con  
números consecutivos"  
        print("Caso 1 (Matriz 3x3): PASSED")  
    except AssertionError as e:  
        print(f"Error: {e}")
```

Packager

python3 mai...

Prueba de sumar diagonal  
Caso 1 (Matriz 3x3): PASSED  
Caso 2 (Matriz 2x2): PASSED  
Caso 3 (Matriz 1x1): PASSED  
Caso 4 (Matriz con números negativos): PASSED  
Caso 5 (Matriz 4x4): PASSED  
Todas las pruebas para sumar\_diagonal\_principal han finaliz  
ado  
Programa realizado por Franz Almanza