

UNIVERSIDAD PRIVADA DOMINGO SAVIO



Documento de Capturas #2

DOCENTE: Jimmy N. Requena Llorentty

TURNO: Mañana

CARRERA: Ing. En Sistemas

MATERIA: Programación II

ESTUDIANTE: Maria Fernanda Vidaurre Alvarado

Fecha y hora actual: 2025-06-24 18:35:42

Santa Cruz- Bolivia

Código #1 Ordenamiento burbuja

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

```
python3 mai... Ask Assistant ✓
Antes: [6, 3, 8, 2, 5]
Después Ordenamiento Burbuja: [2, 3, 5, 6, 8]

--- Ejecutando pruebas con asserts ---
Caso 1 (Lista desordenada): PASSED
Caso 2 (Lista ya ordenada): PASSED
Caso 3 (Lista ordenada a la inversa): PASSED
Caso 4 (Lista con elementos duplicados): PASSED
Caso borde (Lista vacía): PASSED
Caso borde (Lista con un solo elemento): PASSED
/n Código realizado por: Maria F. Vidaurre Alvarado
```

Descripción

La utilidad principal del ordenamiento burbuja en Python, si bien es didáctica, es su simplicidad conceptual. Es fácil de entender e implementar, lo que lo convierte en una excelente herramienta para enseñar los fundamentos de los algoritmos de ordenamiento y cómo funcionan las comparaciones e intercambios de elementos. Aunque ineficiente para grandes conjuntos de datos ($O(n^2)$), puede ser útil en escenarios muy específicos donde la cantidad de elementos es extremadamente pequeña y la legibilidad del código es más crítica que el rendimiento, o cuando se necesita una implementación sencilla sin recurrir a funciones de ordenamiento integradas.

Código #2 Ordenamiento por inserción

```
#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... Ask Assistant ✓

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 Maria F. Vidaurre Alvarad
0
```

Descripción

El ordenamiento por inserción en Python es útil para listas pequeñas o casi ordenadas debido a su simplicidad y eficiencia en estos escenarios. Es un algoritmo "in-place", lo que significa que no requiere memoria adicional significativa. Su estabilidad, manteniendo el orden relativo de elementos iguales, lo hace adecuado para ciertos conjuntos de datos. Aunque su complejidad de tiempo es $O(n^2)$ en el peor y promedio de los casos, lo que lo hace ineficiente para grandes conjuntos de datos desordenados, su facilidad de implementación y su buen rendimiento en casos específicos justifican su uso.

Código #3 Merge Sort

```
1 def merge_sort(lista):
2     # Paso Vencer (Condición Base de la Recursividad):
3     if len(lista) <= 1:
4         return lista
5
6     # Paso 1: DIVIDIR
7     medio = len(lista) // 2
8     mitad_izquierda = lista[:medio]
9     mitad_derecha = lista[medio:]
10
11     # Paso 2: VENCER (Recursión)
12     izquierda_ordenada = merge_sort(mitad_izquierda)
13     derecha_ordenada = merge_sort(mitad_derecha)
14
15     # Paso 3: COMBINAR
16     print(f'Mezclaría {izquierda_ordenada} y {derecha_ordenada}')
17     return merge(izquierda_ordenada, derecha_ordenada)
18
19 def merge(izquierda, derecha):
20     resultado = []
21     i = j = 0
22
23     # Comparar elementos de izquierda y derecha uno por uno
24     while i < len(izquierda) and j < len(derecha):
25         if izquierda[i] < derecha[j]:
26             resultado.append(izquierda[i])
27             i += 1
28         else:
29             resultado.append(derecha[j])
30             j += 1
31     # Agregar los elementos restantes de la lista izquierda o derecha
32     resultado.extend(izquierda[i:])
33     resultado.extend(derecha[j:])
34     return resultado
35
36 # Ejemplo de uso
37 lista = [5, 2, 1, 3, 10, 6, 8, 7, 9, 4, 1, 2, 3, 4, 5, 100, -100, 0, -50, 1.2, 2.5]
38 lista_ordenada = merge_sort(lista)
39 print(f'Lista ordenada: {lista_ordenada}')
40 assert lista_ordenada == sorted(lista), 'El algoritmo no funciona correctamente'
41 print('¡Todas las pruebas con assert pasaron correctamente!')
```

python3 mai... Ask Assistant ✓

Mezclaría [5] y [2]
Mezclaría [1] y [2]
Mezclaría [3] y [1, 2]
Mezclaría [5] y [3]
Mezclaría [10] y [3, 5]
Mezclaría [6] y [2]
Mezclaría [8] y [2, 6]
Mezclaría [3, 5, 10] y [2, 6, 8]
Mezclaría [9] y [7]
Mezclaría [3] y [1]
Mezclaría [5] y [1, 3]
Mezclaría [7, 9] y [1, 3, 5]
Mezclaría [1] y [2]
Mezclaría [4] y [5]
Mezclaría [3] y [4, 5]
Mezclaría [1, 2] y [3, 4, 5]
Mezclaría [4] y [2]
Mezclaría [4] y [1]
Mezclaría [2] y [1, 4]
Mezclaría [2, 4] y [1, 2, 4]
Mezclaría [100] y [-100]
Mezclaría [50] y [-100]
Mezclaría [0] y [-100, 50]
Mezclaría [-50, 100] y [-100, 0, 50]
Mezclaría [1.2] y [3.8]
Mezclaría [2.5] y [1.2, 3.8]
¡Todas las pruebas con assert pasaron correctamente!
Programa realizado por Maria F. Vidaurre Alvarado

Descripción

Merge Sort en Python es útil por su eficiencia y estabilidad. Funciona con el paradigma "divide y vencerás", dividiendo recursivamente la lista hasta que los elementos son individuales y luego fusionándolos de forma ordenada. Su principal ventaja es su complejidad temporal de $O(n \log n)$ en todos los casos (mejor, promedio, peor), lo que lo hace ideal para grandes conjuntos de datos, a diferencia de otros algoritmos que pueden degradarse a $O(n^2)$. Además, es un algoritmo de ordenación estable, manteniendo el orden relativo de elementos iguales.

Código #4 Matrices y matriz en cuadrícula

```
1 #Matrices
2 # 1. Crear la matriz de 3x3
3 teclado = [
4     [1, 2, 3],
5     [4, 5, 6],
6     [7, 8, 9]
7 ]
8
9 # 2. Imprimir la matriz completa
10 print("Matriz original:")
11 for fila in teclado:
12     print(fila)
13
14 # 3. 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 # 4. Modificar el número en la esquina superior izquierda (1 por un 0)
19 teclado[0][0] = 0
20
21 # 5. Imprimir la matriz modificada
22 print("\nMatriz modificada:")
23 for fila in teclado:
```

python3 mai... Ask Assistant

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 Maria F. Vidaurre Alvarado

Descripción

Las matrices en Python son herramientas versátiles que facilitan el manejo de grandes volúmenes de datos numéricos y la ejecución de cálculos matemáticos complejos con alta performance.

Cálculos científicos y de ingeniería: Permiten realizar operaciones de álgebra lineal complejas (multiplicación de matrices, inversión, determinantes) de manera rápida y sencilla, crucial para física, química, y otras disciplinas.

Machine Learning y Data Science: Son la base para representar conjuntos de datos (filas = ejemplos, columnas = características), pesos de redes neuronales, y transformaciones de datos.

Procesamiento de imágenes y gráficos: Las imágenes se representan como matrices de píxeles, y las transformaciones (rotación, escala) se realizan mediante operaciones matriciales.

Código #5 Suma total de elementos de una matriz

```
# Código sumar todo 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}")
```

```
python3 mai... Ask Assistant ✓

--- 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 Maria F. Vidaurre Alvarado

Descripción

Esta función permite calcular la suma de todos los elementos dentro de una matriz o array, o bien, realizar sumas a lo largo de ejes específicos (filas o columnas).
Análisis de datos: Permite obtener rápidamente resúmenes agregados de grandes conjuntos de datos representados como matrices. Por ejemplo, en un dataset donde las filas son observaciones y las columnas son características, se puede calcular la suma total de una característica específica, o la suma de todas las características para una observación particular.

Cálculos estadísticos: Es fundamental para calcular métricas como la suma de cuadrados, la suma de productos, y otras operaciones que forman la base de la estadística descriptiva e inferencial.

Machine Learning y Deep Learning: En el entrenamiento de modelos, a menudo se necesita sumar los valores de los gradientes, los errores, o los pesos de las capas. La función de suma total facilita estas operaciones de manera eficiente.

Procesamiento de imágenes: Las imágenes se representan como matrices de píxeles. Sumar los valores de los píxeles puede ser útil para calcular el brillo total de una imagen o para operaciones de convolución.

Optimización numérica: En algoritmos de optimización, la suma de funciones de costo o penalización es una operación común, y la función de suma total proporciona una forma eficiente de realizarla.

Código #6 Sumar por fila en matriz

```
#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]]
    #print(sumar_por_filas(m1))
```

```
python3 mai... Ask Assistant ✓

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 Maria F. Vidaurre Alvarado
```

Descripción

Es una operación fundamental que permite calcular la suma de todos los elementos a lo largo de una fila específica (o todas las filas) de una matriz o array multidimensional. **Análisis de Datos y Estadísticas:** En conjuntos de datos tabulares (donde las filas representan registros y las columnas características), sumar las filas puede significar calcular el total de ciertos atributos para cada registro. Por ejemplo, si tienes una matriz de ventas diarias por producto, sumar cada fila te daría el total de ventas para cada día.

Machine Learning y Deep Learning:

Normalización y Scaling: A menudo se necesita calcular la suma de los valores en una fila para normalizar los datos, asegurando que cada observación (fila) tenga una escala comparable.

Procesamiento de características: En el pre procesamiento de datos, puedes sumar ciertos atributos dentro de una fila para crear una nueva característica compuesta.

Código #7 Sumar diagonal en matriz

```
1 #Codigo para sumar diagonal en matriz
2 # Definimos la función que suma los elementos de la diagonal principal de una matriz cuadrada
3 def sumar_diagonal_principal(matriz):
4     suma = 0
5     for i in range(len(matriz)):
6         suma += matriz[i][i] # Accede al elemento en la posición (i, i)
7     return suma
8
9 # Función de prueba para verificar que sumar_diagonal_principal funciona correctamente
10 def probar_suma_diagonal_principal():
11     print("\nPrueba de sumar diagonal")
12
13     # Caso 1: matriz 3x3 con números consecutivos
14     m1 = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
15     try:
16         assert sumar_diagonal_principal(m1) == 15, "Fallo en Caso 1: Matriz 3x3 con números consecutivos"
17         print("Caso 1 (Matriz 3x3): PASSED")
18     except AssertionError as e:
19         print(f"Error: {e}")
20
21     # Caso 2: matriz 2x2 con ceros y valores definidos
22     m2 = [[10, 0], [0, 20]]
23     try:
```

python3 mai... Ask Assistant ✓

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 finalizado

Programa realizado por Maria F. Vidaurre Alvarado

Descripción
Es una operación especializada que calcula la suma de los elementos ubicados en la diagonal principal de una matriz cuadrada, o de las diagonales secundarias si se especifica.

Código #8 Transponer matriz (función)

```
#Codigo Transponer Matriz
def transponer_matriz(matriz):
    if not matriz or not matriz[0]:
        return []

    num_filas = len(matriz)
    num_columnas = len(matriz[0])

    # Inicializamos la transpuesta con la estructura correcta
    matriz_transpuesta = []

    for j in range(num_columnas): # Itera sobre las COLUMNAS
        # Itera sobre las FILAS originales
        nueva_fila = []
        for i in range(num_filas):
            nueva_fila.append(matriz[i][j])
        matriz_transpuesta.append(nueva_fila)

    return matriz_transpuesta

# Prueba tu función rigurosamente (incluye matrices no cuadradas):
m1 = [[1, 2, 3], [4, 5, 6]] # 2x3
```

```
python3 mai...
¡Prueba 1 (2x3) pasada! ✓
¡Prueba 2 (matriz 1x1) pasada! ✓
¡Prueba 3 (matriz 1x2) pasada! ✓
¡Prueba 4 (matriz 2x1) pasada! ✓
¡Prueba 5 (matriz vacía) pasada! ✓
¡Prueba 6 (matriz con fila vacía) pasada! ✓
¡Prueba 7 (matriz cuadrada 3x3) pasada! ✓
Todas las pruebas han sido ejecutadas.
```

Codigo realizado por Maria Vidaurre

Descripción

Es una operación fundamental en álgebra lineal que intercambia las filas por las columnas de una matriz. Esto significa que el elemento en la posición (i, j) se mueve a la posición (j, i) .

Código #9 Matriz de Identidad (función)

```
#Codigo Matriz de identidad
def es_identidad(matriz):
    # Requisito 1: Debe ser cuadrada
    num_filas = len(matriz)
    if num_filas == 0:
        return True # Una matriz vacía es trivialmente identidad

    for i in range(num_filas):
        if len(matriz[i]) != num_filas:
            return False # No es cuadrada

    # Requisito 2: Verificar la diagonal y los ceros
    for i in range(num_filas):
        for j in range(num_filas):
            if i == j:
                if matriz[i][j] != 1:
                    return False # La diagonal no tiene 1
            else:
                if matriz[i][j] != 0:
                    return False # Elemento fuera de la diagonal no es 0

    return True # Cumple con todas las condiciones de matriz
```

python3 mai... Ask Assistant 101ms • 14 minutes

¡Prueba 1 (matriz identidad 3x3) pasada! ✓
¡Prueba 2 (matriz 3x3 no identidad por diagonal) pasada! ✓
¡Prueba 3 (matriz no cuadrada) pasada! ✓
¡Prueba 4 (matriz vacía) pasada! ✓
¡Prueba 5 (matriz 1x1 identidad) pasada! ✓
¡Prueba 6 (matriz 1x1 no identidad) pasada! ✓
¡Prueba 7 (matriz con fila vacía) pasada! ✓
¡Prueba 8 (matriz 2x2 con elemento fuera de diagonal no cero) pasada! ✓
Todas las pruebas han sido ejecutadas.

Codigo realizado por Maria Vidaurre

Descripción

Es fundamental por varias razones en el ámbito de la manipulación matricial y el álgebra lineal. Una matriz identidad es una matriz cuadrada donde todos los elementos de la diagonal principal son uno, y todos los demás elementos son cero.

Su utilidad principal radica en ser el elemento neutro para la multiplicación de matrices. Al igual que el número 1 no altera un número al multiplicarlo ($x \cdot 1 = x$), la matriz identidad no altera una matriz cuando se multiplica por ella ($A \cdot I = A$ y $I \cdot A = A$).

Código #10 Matriz simétrica (función)

```
#Codigo Matriz simetrica
def es_simetrica(matriz):
    # Requisito 1: Debe ser cuadrada
    num_filas = len(matriz)
    if num_filas == 0:
        return True # Una matriz vacía es trivialmente simétrica

    for i in range(num_filas):
        if len(matriz[i]) != num_filas:
            return False # No es cuadrada

    # Requisito 2: Comparar matriz[i][j] con matriz[j][i]
    for i in range(num_filas):
        for j in range(i + 1, num_filas): # Solo necesitamos
            # chequear la triangular superior
            if matriz[i][j] != matriz[j][i]:
                return False # ¡Con una diferencia es suficiente!

    return True # Si nunca encontramos diferencias, es simétrica

# Prueba tu función:
sim = [[1, 7, 3], [7, 4, -5], [3, -5, 6]]
no_sim = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

```
python3 mai... Ask Assistant 94ms • 1
¡Prueba 1 (matriz simétrica 3x3) pasada! ✓
¡Prueba 2 (matriz 3x3 no simétrica) pasada! ✓
¡Prueba 3 (matriz no cuadrada) pasada! ✓
¡Prueba 4 (matriz vacía) pasada! ✓
¡Prueba 5 (matriz 1x1) pasada! ✓
¡Prueba 6 (matriz 2x2 no simétrica) pasada! ✓
¡Prueba 7 (matriz con fila vacía que no es cuadrada) pasada! ✓

Codigo realizado por Maria Vidaurre
```

Descripción

Tiene una utilidad inmensa porque sus propiedades algebraicas y geométricas simplifican el análisis y la resolución de problemas complejos en una amplia gama de disciplinas científicas y de ingeniería. Permite la diagonalización, la interpretación de valores propios como cantidades físicas reales y la simplificación de formas cuadráticas, lo que se traduce en herramientas poderosas para el análisis de datos, el modelado físico y la optimización.

Código #11 Sala de Cine

```
1 #Codigo Sala Cine
2 # Crear la sala con precios asignados
3 def crear_sala(filas, columnas):
4     sala = []
5     for i in range(filas):
6         fila = []
7         for j in range(columnas):
8             # Asigna un precio según la ubicación (ejemplo simple)
9             if 2 <= j <= 5:
10                 precio = 50 # Asientos centrales
11             else:
12                 precio = 30 # Asientos de los costados
13             fila.append({"estado": "L", "precio": precio})
14         sala.append(fila)
15     return sala
16
17 # Mostrar la sala con precios y estados
18 def mostrar_sala(sala):
19     print("\n      " + " ".join(f"{j:^5}" for j in
20 range(len(sala[0])))
21     print("      " + " ".join("-" * 5 for _ in range(len(sala[0])))
22     for i, fila in enumerate(sala):
23         estado_fila = " ".join(f"{a['estado']:^5}" for a in fila)
```

python3 mai... Ask Assistant 16s

Menú:
1. Ocupar asiento individual
2. Buscar y ocupar N asientos juntos
0. Salir
Elige una opción: 1
Fila: 1
Columna: 1
Asiento (1, 1) reservado por Bs. 30

Sala actual:

	0	1	2	3	4	5	6	7
F 0	L	L	L	L	L	L	L	L
F 1	L	O	L	L	L	L	L	L
F 2	L	L	L	L	L	L	L	L
F 3	L	L	L	L	L	L	L	L
F 4	L	L	L	L	L	L	L	L

Asientos libres: 39

Menú:
1. Ocupar asiento individual
2. Buscar y ocupar N asientos juntos
0. Salir
Elige una opción: 0
Gracias por usar el sistema de reserva de cine.

Codigo realizado por Maria Vidaurre

Descripción
Funciona como una herramienta estratégica que mejora la eficiencia operativa, reduce costos, optimiza la capacidad de la sala y, lo más importante, eleva significativamente la experiencia del cliente.

Código #12 Diccionario

```
#Codigo Diccionario
#Diccionario cancion
cancion = {
    "titulo": "Locos",
    "artista": "León Larregui",
    "album": "Voluma",
    "duracion_segundos": 177,
    "genero": "Alternative Rock",
    "fecha_lanzamiento": "01-01-2016"
}
print("Información de la canción")
print(f"Título: {cancion['titulo']}")
print(f"Artista: {cancion['artista']}")
print(f"Álbum: {cancion['album']}")
print(f"Duración: {cancion['duracion_segundos']} segundos")
print(f"Género: {cancion['genero']}")
print(f"Fecha de lanzamiento: {cancion['fecha_lanzamiento']}")

print("\nPrograma realizado por Maria F. Vidaurre Alvarado")
```

Información de la canción
Título: Locos
Artista: León Larregui
Álbum: Voluma
Duración: 177 segundos
Género: Alternative Rock
Fecha de lanzamiento: 01-01-2016

Programa realizado por Maria F. Vidaurre Alvarado

Descripción

Los diccionarios son esenciales para organizar y manipular datos de forma eficiente mediante la asociación de claves con valores.

Cada palabra es una clave (única).

La definición de esa palabra es su valor.

Código #13 Inventario

```
#Codigo Inventario
# 1. Crear una lista vacía llamada inventario
inventario = []

# 2. Crear al menos tres diccionarios de productos diferentes
producto1 = {
    "nombre": "Chocolate para Taza 'El Ceibo'",
    "stock": 50
}

producto2 = {
    "nombre": "Café de los Yungas",
    "stock": 100
}

producto3 = {
    "nombre": "Quinua Real en Grano",
    "stock": 80
}

# 3. Añadir los productos al inventario
inventario.append(producto1)
inventario.append(producto2)
```

```
python3 mai...
→ stock
→ proveedor

Valores del diccionario producto:
→ P001
→ Chocolate para Taza 'El Ceibo'
→ 15.5
→ 50
→ El Ceibo Ltda.

Contenido completo del diccionario producto:
codigo: P001
nombre: Chocolate para Taza 'El Ceibo'
precio_unitario: 15.5
stock: 50
proveedor: El Ceibo Ltda.

❌ La clave 'en_oferta' no existe.
Stock disponible: 50 unidades

--- Detalle de productos usando .items() ---
nombre → Chocolate para Taza 'El Ceibo'
stock → 50
nombre → Café de los Yungas
stock → 100
nombre → Quinua Real en Grano
stock → 80

Codigo realizado por Maria Vidaurre
```

Descripción

Puede ser una herramienta poderosa que transforma la gestión manual y propensa a errores en un proceso automatizado, preciso y basado en datos. Esto no solo ahorra tiempo y dinero, sino que también proporciona una visión clara del estado del negocio, permitiendo tomar decisiones más inteligentes y ofrecer un mejor servicio al cliente.

Código #14 To Do List

```
##Codigo To Do List
# 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!")
```

```
python3 mai...
3. Marcar tarea como completada
4. Eliminar tarea
0. Salir
Elige una opción: 1
Descripción de la nueva tarea: tarea obligatoria
Prioridad (alta, media, baja): alta
✅ Tarea 'tarea obligatoria' 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 | tarea obligatoria (Prioridad: alta)
```

```
==== 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: 0
¡Hasta pronto!
```

Codigo realizado por Maria Vidaurre

Descripción

Crear una To-Do List en Python es un proyecto excelente para organizar tareas y aprender programación. Te permite aplicar conceptos básicos como estructuras de datos, funciones y manejo de archivos. Es una forma práctica de mejorar tus habilidades de codificación mientras desarrollas una herramienta personalizable y útil para tu día a día, gestionando tus pendientes eficientemente.

Código #15 Batalla Naval

```
##Codigo Batalla naval
import random

FILAS = 4
COLUMNAS = 2
BARCOS = 3

# Crear un tablero vacío
def crear_tablero():
    return [[0 for _ in range(COLUMNAS)] for _ in range(FILAS)]

# Mostrar el tablero
def mostrar_tablero(tablero, ocultar_barcos=False):
    print(" " + " ".join(str(i + 1) for i in range(COLUMNAS)))
    for i, fila in enumerate(tablero):
        letra = chr(ord('A') + i)
        fila_mostrar = []
        for celda in fila:
            if ocultar_barcos and celda == 1:
                fila_mostrar.append("0")
            elif celda == 0:
                fila_mostrar.append("0")
            elif celda == 1:
                fila_mostrar.append("X")

        print(letra + " " + " ".join(fila_mostrar))

# Ejemplo de uso
tablero = crear_tablero()
mostrar_tablero(tablero)

# Simulación de juego
def jugar():
    # Turno 1
    print("\n--- Turno 1 ---")
    print("Tu tablero:")
    mostrar_tablero(tablero)
    print("Tus disparos:")
    disparos = input("Dispara (ej. A3): ")
    # Simulación de disparo
    if disparos == "C2":
        print("Maria disparó al agua.")
    else:
        print("CPU dispara a D2")
        print("CPU disparó al agua.")

    # Turno 2
    print("\n--- Turno 2 ---")
    print("Tu tablero:")
    mostrar_tablero(tablero)
    print("Tus disparos:")
    disparos = input("Dispara (ej. A3): ")
    # Simulación de disparo
    if disparos == "D1":
        print("Maria hizo ¡Tocado!")
    else:
        print("CPU dispara a C2")
        print("CPU hizo ¡Tocado!")
        print("¡La CPU gana!")

# Ejecución
jugar()
```

Descripción

Es una excelente herramienta educativa para aprender y aplicar conceptos fundamentales de programación, desde estructuras de datos básicas hasta algoritmos más complejos de IA. Al mismo tiempo, ofrece una forma accesible y divertida de disfrutar de un juego clásico que agudiza el pensamiento estratégico y la lógica.

Código #16 Gestor de contactos (agenda)

##Codigo Agenda

```
agenda = {}
```

```
def agregar_contacto(nombre, **detalles):
```

```
    """
```

```
    Agrega un nuevo contacto a la agenda con detalles flexibles.
```

```
    'detalles' es un diccionario que puede contener 'telefonos',
```

```
'email', 'direccion', etc.
```

```
    """
```

```
    if nombre in agenda:
```

```
        print(f"El contacto '{nombre}' ya existe.")
```

```
        return False
```

```
    agenda[nombre] = detalles
```

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

```
    return True
```

```
def buscar_por_nombre(nombre):
```

```
    """
```

```
    Busca y devuelve la información del contacto por nombre.
```

```
    """
```

```
    return agenda.get(nombre, None)
```

```
def editar_contacto(nombre, **nuevos_detalles):
```

python3 mai...

Ask Assistant

51s • 14 minutes ag

```
Introduce un numero de telefono (o 'fin' para terminar): fin
```

```
Introduce un tipo de detalle (ej. 'telefono', 'email', 'direccion', o 'fin'
```

```
para terminar): fin
```

```
Contacto 'Maria F' agregado.
```

```
--- Menú de la Agenda ---
```

```
1. Añadir contacto
```

```
2. Ver contacto
```

```
3. Editar contacto
```

```
4. Eliminar contacto
```

```
5. Listar todos los contactos
```

```
6. Salir
```

```
Selecciona una opción: 5
```

```
--- Todos los Contactos ---
```

```
- Maria F
```

```
-----
```

```
--- Menú de la Agenda ---
```

```
1. Añadir contacto
```

```
2. Ver contacto
```

```
3. Editar contacto
```

```
4. Eliminar contacto
```

```
5. Listar todos los contactos
```

```
6. Salir
```

```
Selecciona una opción: 6
```

```
Saliendo de la agenda. ¡Hasta pronto!
```

```
Codigo realizado por Maria Vidaurre
```

```
□
```

Python Diff

Ln 75, Col 1 • Spaces: 4 History

Descripción

Un gestor de contactos (agenda) en Python es una herramienta útil y versátil para organizar información personal o profesional. Permite a los usuarios almacenar, editar, buscar y eliminar datos de contacto como nombres, números de teléfono, correos electrónicos y direcciones.

Código #17 Matriz Teclado

```
#Matriz Teclado
# 1. Declaramos una matriz que simula un teclado
teclado = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9],
    ["*", 0, "#"]
]

# 2. Imprimimos la matriz como cuadrícula
print("► MATRIZ DEL TECLADO:\n")
for fila in teclado:
    for elemento in fila:
        # Imprime cada elemento con tabulación, sin salto de línea
        print(elemento, end="\t")
    # Al final de cada fila, hacemos un salto de línea
    print()

# 3. Crear una matriz 5x5 de ceros usando bucles anidados
print("\n► MATRIZ 5x5 CON CEROS (usando bucles):\n")
matriz_5x5 = [] # Lista vacía para la matriz
```

python3 mai... Ask Assistant ✓

► MATRIZ DEL TECLADO:

1	2	3
4	5	6
7	8	9
*	0	#

► MATRIZ 5x5 CON CEROS (usando 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

► MATRIZ 5x5 CON CEROS (usando comprensión de listas):

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 Maria Vidaurre

Descripción

Representa un teclado telefónico como una matriz, ilustrando cómo se pueden modelar cuadrículas del mundo real. Luego, presenta dos métodos para crear una matriz de 5x5 llena de ceros: primero, usando bucles anidados tradicionales para una comprensión paso a paso.

Código #18 Suma Diagonal Secundaria

```
#Codigo suma diagonal secundaria
def sumar_diagonal_secundaria(matriz):
```

```
    """
    Recibe una matriz cuadrada (misma cantidad de filas y columnas)
    y devuelve la suma de los elementos en la diagonal secundaria.
    La diagonal secundaria va desde la esquina superior derecha
    hasta la esquina inferior izquierda.
```

```
    Por ejemplo, en una matriz 3x3:
```

```
    [[a, b, c],
```

```
     [d, e, f],
```

```
     [g, h, i]]
```

```
    La diagonal secundaria está en las posiciones: (0,2), (1,1), (2,0)
```

```
    y su suma sería: c + e + g
```

```
    """
```

```
    n = len(matriz) # Número de filas (y columnas, ya que es cuadrada)
```

```
    suma = 0
```

```
    for i in range(n):
```

```
        suma += matriz[i][n - 1 - i] # Accede al elemento en la posición (i, n-1-i)
```

```
    return suma
```

Probando sumar_diagonal_secundaria...

Matriz original (m1):

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

Suma de la diagonal secundaria para m1: 15

Matriz original (m2):

```
[[10, 1], [2, 20]]
```

Suma de la diagonal secundaria para m2: 3

Matriz original (m3):

```
[[42]]
```

Suma de la diagonal secundaria para m3: 42

¡Pruebas para sumar_diagonal_secundaria pasaron!

Programa realizado por Maria Vidaurre

Descripción

Es una operación especializada que calcula la suma de los elementos ubicados en la diagonal secundaria de una matriz cuadrada, o de las diagonales secundarias si se especifica.

Código #19 Sala Cine Pro

```
#Codigo sala de cine
# Crear la sala con precios asignados
def crear_sala(filas, columnas):
    sala = []
    for i in range(filas):
        fila = []
        for j in range(columnas):
            # Asigna un precio según la ubicación (ejemplo simple)
            if 2 <= j <= 5:
                precio = 50 # Asientos centrales
            else:
                precio = 30 # Asientos de los costados
            fila.append({"estado": "L", "precio": precio})
        sala.append(fila)
    return sala

# Mostrar la sala con precios y estados
def mostrar_sala(sala):
    print("\n      " + " ".join(f"{j:^5}" for j in range(len(sala[0]))))
    print("      " + " ".join("-" * 5 for _ in range(len(sala[0]))))
    for i, fila in enumerate(sala):
        estado_fila = " ".join(f"{a['estado']:^5}" for a in fila)
        print(f"F{i>2} | {estado_fila}")
```

```
python3 mai... Ask Assistant ✓
Columna: 4
Asiento (4, 4) reservado por Bs. 50

Sala actual:

    0    1    2    3    4    5    6    7
    --    --    --    --    --    --    --
F 0 | L    L    L    L    L    L    L    L
F 1 | L    L    L    L    L    L    L    L
F 2 | L    L    L    L    L    L    L    L
F 3 | L    L    L    L    L    L    L    L
F 4 | L    L    L    L    0    L    L    L

Asientos libres: 39

Menú:
1. Ocupar asiento individual
2. Buscar y ocupar N asientos juntos
0. Salir
Elige una opción: 0
Gracias por usar el sistema de reserva de cine. 🍿

Programa realizado por Maria Vidaurre
```

Descripción

Funciona como una herramienta estratégica que mejora la eficiencia operativa, reduce costos, optimiza la capacidad de la sala y, lo más importante, eleva significativamente la experiencia del cliente.