CLASE 8: ESTRUCTURAS DE DATOS

Listas, Tuplas, Diccionarios y Sets en Python

Contenido:

PARTE 1: LISTAS

1.1 Que es una Lista

Una **lista** es una coleccion ordenada y modificable (mutable) de elementos. Es la estructura de datos mas versatil en Python.

Caracteristicas:

- **Ordenada**: Los elementos mantienen su posicion (indices)
- **Mutable**: Se puede modificar despues de crearla
- **Permite duplicados**: Puede tener elementos repetidos
- **Heterogenea**: Puede contener diferentes tipos de datos

```
# Crear listas
frutas = ["manzana", "naranja", "platano"]
numeros = [1, 2, 3, 4, 5]
mixta = [42, "texto", True, 3.14]
vacia = []

# Indices: positivos desde 0, negativos desde -1
print(frutas[0])     # "manzana" (primero)
print(frutas[-1])    # "platano" (ultimo)
```

1.2 Operaciones Basicas con Listas

```
# Modificar elementos
frutas[1] = "fresa"

# Agregar elementos
frutas.append("uva")           # Al final
frutas.insert(0, "melon")      # En posicion especifica
frutas.extend(["kiwi", "pera"]) # Agregar varios

# Eliminar elementos
frutas.remove("fresa")         # Por valor
eliminado = frutas.pop(0)      # Por indice (retorna el valor)
frutas.clear()                 # Vaciar lista

# Slicing
nums = [0, 1, 2, 3, 4, 5]
print(nums[1:4])    # [1, 2, 3]
print(nums[:3])     # [0, 1, 2]
print(nums[3:])     # [3, 4, 5]
print(nums[::-1])   # [5, 4, 3, 2, 1, 0] (invertida)
```

1.3 Metodos Importantes de Listas

```
nums = [3, 1, 4, 1, 5, 9, 2, 6]

# Busqueda
print(4 in nums)         # True
```

```
print(nums.index(5))     # 4 (posicion)
print(nums.count(1))     # 2 (apariciones)

# Ordenamiento
nums.sort()              # Ordena la lista (modifica)
nums.sort(reverse=True)  # Descendente
ordenada = sorted(nums)  # Nueva lista ordenada (no modifica)

# Otros
nums.reverse()           # Invierte orden
copia = nums.copy()      # Copia independiente
print(len(nums))         # Cantidad de elementos
print(sum(nums))         # Suma (solo numeros)
print(min(nums), max(nums))  # Minimo y maximo
```

PARTE 2: TUPLAS

2.1 Que es una Tupla

Una **tupla** es una coleccion ordenada e INMUTABLE de elementos. Una vez creada, no se puede modificar.

Caracteristicas:

- **Ordenada**: Los elementos mantienen su posicion
- **Inmutable**: NO se puede modificar despues de crearla
- **Permite duplicados**: Puede tener elementos repetidos
- **Mas eficiente**: Usa menos memoria que las listas

```
# Crear tuplas
coordenada = (10, 20)
colores = ("rojo", "verde", "azul")
mixta = (1, "texto", True)
vacia = ()
un_elemento = (42,)  # IMPORTANTE: la coma es necesaria

# Acceso igual que listas
print(colores[0])     # "rojo"
print(colores[-1])    # "azul"
print(colores[0:2])   # ("rojo", "verde")
```

2.2 Operaciones con Tuplas

```
# Las tuplas NO se pueden modificar
coordenada = (10, 20)
# coordenada[0] = 15  # ERROR! TypeError

# Pero se pueden reasignar completamente
coordenada = (15, 25)  # OK, nueva tupla

# Desempaquetado de tuplas
punto = (100, 200, 300)
x, y, z = punto
print(x, y, z)  # 100 200 300

# Intercambio de variables (Python trick)
a, b = 10, 20
a, b = b, a  # Intercambio usando tuplas
print(a, b)  # 20 10

# Concatenar tuplas (crea nueva)
t1 = (1, 2, 3)
t2 = (4, 5, 6)
t3 = t1 + t2  # (1, 2, 3, 4, 5, 6)
```

2.3 Metodos de Tuplas

Las tuplas tienen solo 2 metodos (por ser inmutables):

```
colores = ("rojo", "verde", "azul", "rojo", "rojo")

# count() - contar apariciones
print(colores.count("rojo"))  # 3

# index() - encontrar posicion
print(colores.index("verde"))  # 1

# Funciones utiles
print(len(colores))   # 5
print("azul" in colores)  # True
print(min((1, 2, 3)))  # 1
print(max((1, 2, 3)))  # 3
```

TIP: Usa tuplas para datos que no deben cambiar: coordenadas, fechas, colores RGB, etc.

PARTE 3: DICCIONARIOS

3.1 Que es un Diccionario

Un **diccionario** es una coleccion de pares clave:valor. Permite acceder a los valores mediante claves unicas en lugar de indices.

Caracteristicas:

- **Pares clave:valor**: Cada elemento tiene una clave y un valor
- **Claves unicas**: No puede haber claves duplicadas
- **Mutable**: Se puede modificar despues de crearlo
- **Ordenado**: Desde Python 3.7, mantiene orden de insercion

```
# Crear diccionarios
persona = {
    "nombre": "Ana",
    "edad": 25,
    "ciudad": "Madrid"
}
vacio = {}

# Acceder a valores
print(persona["nombre"])      # "Ana"
print(persona.get("edad"))    # 25
print(persona.get("pais", "No especificado"))  # Valor por defecto
```

3.2 Operaciones con Diccionarios

```
persona = {"nombre": "Ana", "edad": 25}

# Agregar/Modificar
persona["ciudad"] = "Madrid"     # Agregar nuevo
persona["edad"] = 26             # Modificar existente
persona.update({"pais": "Espana", "edad": 27})  # Varios a la vez

# Eliminar
del persona["ciudad"]            # Por clave
edad = persona.pop("edad")       # Eliminar y obtener valor
persona.clear()                  # Vaciar todo

# Verificar existencia
print("nombre" in persona)       # True (busca en claves)
print("Ana" in persona.values()) # True (busca en valores)
```

3.3 Metodos Importantes de Diccionarios

```
persona = {"nombre": "Ana", "edad": 25, "ciudad": "Madrid"}

# Obtener claves, valores y pares
print(persona.keys())    # dict_keys(['nombre', 'edad', 'ciudad'])
print(persona.values())  # dict_values(['Ana', 25, 'Madrid'])
```

```
print(persona.items())   # dict_items([('nombre', 'Ana'), ...])

# Iterar sobre diccionario
for clave in persona:
    print(f"{clave}: {persona[clave]}")

# Mejor forma de iterar
for clave, valor in persona.items():
    print(f"{clave}: {valor}")

# Copiar diccionario
copia = persona.copy()
```

ERROR COMUN: Acceder a una clave que no existe con [] causa KeyError. Usa .get() para evitarlo.

PARTE 4: SETS (CONJUNTOS)

4.1 Que es un Set

Un **set** es una coleccion NO ordenada de elementos UNICOS. Automaticamente elimina duplicados.

Caracteristicas:

- **No ordenado**: Los elementos no tienen posicion fija
- **Elementos unicos**: No permite duplicados
- **Mutable**: Se puede modificar (agregar/eliminar)
- **No indexable**: No se puede acceder por indice

```
# Crear sets
numeros = {1, 2, 3, 4, 5}
letras = set("abracadabra")  # {'a', 'b', 'r', 'c', 'd'}
desde_lista = set([1, 2, 2, 3, 3, 3])  # {1, 2, 3}
vacio = set()  # IMPORTANTE: {} crea diccionario vacio

# Los duplicados se eliminan automaticamente
nums = {1, 2, 2, 3, 3, 3}
print(nums)  # {1, 2, 3}
```

4.2 Operaciones con Sets

```
nums = {1, 2, 3}

# Agregar elementos
nums.add(4)           # Agregar uno
nums.update([5, 6])   # Agregar varios

# Eliminar elementos
nums.remove(3)        # Eliminar (error si no existe)
nums.discard(99)      # Eliminar (NO da error si no existe)
elemento = nums.pop() # Eliminar aleatorio
nums.clear()          # Vaciar

# Verificar existencia
print(3 in nums)      # True/False
```

4.3 Operaciones de Conjuntos

```
A = {1, 2, 3, 4}
B = {3, 4, 5, 6}

# Union: elementos en A o B (o ambos)
print(A | B)           # {1, 2, 3, 4, 5, 6}
print(A.union(B))      # {1, 2, 3, 4, 5, 6}

# Interseccion: elementos en ambos
print(A & B)           # {3, 4}
print(A.intersection(B)) # {3, 4}
```

```
# Diferencia: elementos en A pero no en B
print(A - B)           # {1, 2}
print(A.difference(B)) # {1, 2}

# Diferencia simetrica: elementos en A o B pero no en ambos
print(A ^ B)           # {1, 2, 5, 6}
print(A.symmetric_difference(B))  # {1, 2, 5, 6}

# Subconjuntos y superconjuntos
print({1, 2}.issubset(A))    # True
print(A.issuperset({1, 2}))  # True
```

TIP: Usa sets cuando necesites eliminar duplicados o verificar pertenencia rapidamente.

PARTE 5: TABLA COMPARATIVA

Comparacion de Estructuras de Datos

Caracteristica	Lista	Tupla	Diccionario	Set
Sintaxis	[ ]	( )	{ clave:valor }	{ }
Ordenada	Si	Si	Si (3.7+)	No
Mutable	Si	No	Si	Si
Duplicados	Si	Si	Claves: No	No
Indexable	Si	Si	Por clave	No
Agregar	append()	-	[ ]= o update()	add()
Eliminar	remove()	-	del o pop()	remove()
Buscar	in, index()	in, index()	in, get()	in
Uso tipico	Colecciones	Datos fijos	Mapeos	Unicos

Tabla de Metodos Principales

Accion	Lista	Tupla	Diccionario	Set
Crear vacio	[] o list()	() o tuple()	{} o dict()	set()
Longitud	len()	len()	len()	len()
Agregar 1	append(x)	-	d[k]=v	add(x)
Agregar varios	extend()	-	update()	update()
Eliminar valor	remove(x)	-	pop(k)	remove(x)
Eliminar indice	pop(i)	-	del d[k]	discard(x)
Buscar	x in lista	x in tupla	k in dict	x in set
Contar	count(x)	count(x)	-	-
Ordenar	sort()	-	-	-
Copiar	copy()	-	copy()	copy()

PARTE 6: CUANDO USAR CADA ESTRUCTURA

Lista - Usa cuando:

- Necesitas una coleccion ordenada de elementos
- Vas a modificar, agregar o eliminar elementos frecuentemente
- Necesitas acceder a elementos por posicion (indice)
- Los duplicados son validos/esperados

Ejemplos: Lista de tareas, historial de acciones, cola de espera

Tupla - Usa cuando:

- Los datos no deben cambiar despues de crearse
- Necesitas una estructura mas eficiente en memoria
- Quieres usar como clave de diccionario (las listas no pueden)
- Retornas multiples valores de una funcion

Ejemplos: Coordenadas (x, y), fechas, colores RGB, configuraciones fijas

Diccionario - Usa cuando:

- Necesitas asociar claves con valores (mapeo)
- Quieres acceso rapido por nombre/clave en lugar de posicion
- Manejas datos estructurados con campos nombrados
- Necesitas contar ocurrencias o agrupar datos

Ejemplos: Perfil de usuario, configuraciones, contador de palabras

Set - Usa cuando:

- Necesitas eliminar duplicados automaticamente
- Vas a hacer operaciones de conjuntos (union, interseccion)
- Solo te importa si un elemento existe o no (membresia)
- El orden no importa

Ejemplos: Tags unicos, usuarios conectados, palabras unicas en texto

Diagrama de Decision

```
Preguntate:

1. Necesito pares clave:valor? --> DICCIONARIO

2. Los elementos deben ser unicos?
   - Si, y el orden no importa --> SET
```

   - Si, pero necesito orden --> Lista + eliminar duplicados

3. Los datos deben ser inmutables?
   - Si --> TUPLA

4. Necesito modificar la coleccion?
   - Si --> LISTA
   - No --> TUPLA (mas eficiente)

5. Por defecto, para colecciones generales --> LISTA

PARTE 7: EJERCICIOS DE CLASE RESUELTOS

Ejercicios de Listas

Ejercicio 1: Operaciones basicas con listas

```
peliculas = ["Inception", "Matrix", "Interstellar"]
peliculas.append("Avatar")
peliculas.insert(0, "Titanic")
print(peliculas)  # ['Titanic', 'Inception', 'Matrix', 'Interstellar', 'Avatar']
print(peliculas[2])   # 'Matrix'
print(peliculas[-1])  # 'Avatar'
print(peliculas[1:4]) # ['Inception', 'Matrix', 'Interstellar']
```

Ejercicio 2: Estadisticas de lista

```
calificaciones = [85, 90, 78, 92, 88, 75, 95]
print(f"Total: {len(calificaciones)}")      # 7
print(f"Suma: {sum(calificaciones)}")       # 603
print(f"Promedio: {sum(calificaciones)/len(calificaciones):.2f}")  # 86.14
print(f"Maxima: {max(calificaciones)}")     # 95
print(f"Minima: {min(calificaciones)}")     # 75
```

Ejercicios de Tuplas

Ejercicio 3: Trabajando con tuplas

```
# Coordenadas de un punto
punto = (100, 200)
x, y = punto  # Desempaquetado
print(f"X: {x}, Y: {y}")

# Tupla de colores RGB
rojo = (255, 0, 0)
verde = (0, 255, 0)
azul = (0, 0, 255)

# Intercambio de variables
a, b = 10, 20
a, b = b, a
print(f"a={a}, b={b}")  # a=20, b=10
```

Ejercicios de Diccionarios

Ejercicio 4: Gestion de estudiantes

```
estudiante = {
    "nombre": "Ana Garcia",
    "edad": 20,
    "carrera": "Ingenieria",
    "promedio": 8.5
}
# Acceder y modificar
```

```
print(estudiante["nombre"])
estudiante["promedio"] = 8.7
estudiante["semestre"] = 4  # Agregar nuevo

# Iterar
for clave, valor in estudiante.items():
    print(f"{clave}: {valor}")
```

Ejercicio 5: Contador de palabras

```
texto = "el gato y el perro y el pajaro"
palabras = texto.split()
contador = {}

for palabra in palabras:
    if palabra in contador:
        contador[palabra] += 1
    else:
        contador[palabra] = 1

print(contador)  # {'el': 3, 'gato': 1, 'y': 2, 'perro': 1, 'pajaro': 1}
```

Ejercicios de Sets

Ejercicio 6: Eliminar duplicados

```
numeros = [1, 2, 2, 3, 3, 3, 4, 4, 4, 4]
unicos = list(set(numeros))
print(unicos)  # [1, 2, 3, 4]

# Contar elementos unicos
print(f"Elementos unicos: {len(set(numeros))}")  # 4
```

Ejercicio 7: Operaciones de conjuntos

```
estudiantes_python = {"Ana", "Luis", "Maria", "Pedro"}
estudiantes_java = {"Maria", "Pedro", "Carlos", "Sofia"}

# Estudiantes en ambos cursos
ambos = estudiantes_python & estudiantes_java
print(f"En ambos: {ambos}")  # {'Maria', 'Pedro'}

# Estudiantes en algun curso
todos = estudiantes_python | estudiantes_java
print(f"Todos: {todos}")

# Solo Python
solo_python = estudiantes_python - estudiantes_java
print(f"Solo Python: {solo_python}")  # {'Ana', 'Luis'}
```

Ejercicio Integrador: Sistema de Inventario

```
# Sistema que usa las 4 estructuras
inventario = {}  # Diccionario: producto -> cantidad

# Lista de operaciones realizadas
historial = []

# Set de categorias unicas
categorias = set()

# Tuplas para productos con precio fijo
PRODUCTOS = (
    ("laptop", 999.99, "electronica"),
    ("mouse", 29.99, "electronica"),
    ("libro", 19.99, "papeleria"),
)

# Agregar productos al inventario
for nombre, precio, categoria in PRODUCTOS:
    inventario[nombre] = {"precio": precio, "cantidad": 0, "categoria": categoria}
    categorias.add(categoria)
    historial.append(f"Producto {nombre} registrado")

# Agregar stock
inventario["laptop"]["cantidad"] = 10
historial.append("Stock de laptop actualizado a 10")

# Mostrar categorias unicas
```

```
print(f"Categorias: {categorias}")  # {'electronica', 'papeleria'}

# Mostrar historial
print("Ultimas operaciones:")
for op in historial[-3:]:
    print(f"  - {op}")
```

PARTE 8: EJERCICIOS ADICIONALES

NIVEL PRINCIPIANTE

Ejercicio P1: Lista de Compras

Crea una lista de compras con 5 productos. Agrega 2 mas con append(), elimina uno con remove(), y muestra la lista final.

Ejercicio P2: Tupla de Fecha

Crea una tupla con tu fecha de nacimiento (dia, mes, anio). Desempaquetala en 3 variables e imprime en formato 'Naci el dia X del mes Y del anio Z'.

Ejercicio P3: Diccionario Personal

Crea un diccionario con tu informacion: nombre, edad, ciudad, hobby. Agrega una nueva clave 'email' y modifica la edad. Imprime todas las claves y valores.

Ejercicio P4: Set de Numeros

Dada la lista [1,2,2,3,3,3,4,4,4,4,5,5,5,5,5], usa un set para obtener los numeros unicos y cuenta cuantos son.

Ejercicio P5: Conversion entre Estructuras

Crea una lista ['a','b','c'], conviertela a tupla, luego a set. Imprime el tipo de cada resultado.

NIVEL INTERMEDIO

Ejercicio I1: Agenda de Contactos

Crea un diccionario de contactos donde la clave es el nombre y el valor es otro diccionario con telefono y email. Implementa funciones para: agregar contacto, buscar contacto, listar todos.

Ejercicio I2: Analisis de Texto con Sets

Dados dos textos, usa sets para encontrar: palabras comunes, palabras unicas del primer texto, total de palabras unicas en ambos.

Ejercicio I3: Lista de Tuplas Ordenada

Crea una lista de tuplas con (nombre, calificacion) de 5 estudiantes. Ordena la lista por calificacion de mayor a menor.

Ejercicio I4: Diccionario de Frecuencias

Dado un texto, crea un diccionario que cuente la frecuencia de cada letra (ignorando espacios y mayusculas).

Ejercicio I5: Combinando Estructuras

Crea un sistema de biblioteca: diccionario de libros (titulo -> autor, anio, disponible), set de generos, lista de prestamos (tuplas con libro, usuario, fecha).

NIVEL AVANZADO

Ejercicio A1: Sistema de Votacion

Implementa un sistema de votacion con: lista de candidatos, diccionario de votos, set de votantes (para evitar votos duplicados). Funciones para: votar, ver resultados, declarar ganador.

Ejercicio A2: Inventario de Tienda

Crea un sistema de inventario con diccionario de productos (cada producto tiene: nombre, precio, cantidad, categoria). Implementa: agregar producto, vender, reponer stock, productos por categoria, productos bajo stock.

Ejercicio A3: Red Social Basica

Modela usuarios como diccionarios con: nombre, amigos (set), publicaciones (lista de tuplas con texto y fecha). Implementa: agregar amigo, publicar, amigos en comun entre dos usuarios.

SOLUCIONES EJERCICIOS ADICIONALES

SOLUCIONES NIVEL PRINCIPIANTE

Solucion P1: Lista de Compras

```
compras = ["leche", "pan", "huevos", "queso", "jamon"]
compras.append("yogurt")
compras.append("cereal")
compras.remove("queso")
print(compras)  # ['leche', 'pan', 'huevos', 'jamon', 'yogurt', 'cereal']
```

Solucion P2: Tupla de Fecha

```
nacimiento = (15, 6, 1995)
dia, mes, anio = nacimiento
print(f"Naci el dia {dia} del mes {mes} del anio {anio}")
```

Solucion P3: Diccionario Personal

```
persona = {
    "nombre": "Juan",
    "edad": 25,
    "ciudad": "Mexico",
    "hobby": "programar"
}
persona["email"] = "juan@email.com"
persona["edad"] = 26
for clave, valor in persona.items():
    print(f"{clave}: {valor}")
```

Solucion P4: Set de Numeros

```
lista = [1,2,2,3,3,3,4,4,4,4,5,5,5,5,5]
unicos = set(lista)
print(f"Numeros unicos: {unicos}")
print(f"Cantidad: {len(unicos)}")  # 5
```

Solucion P5: Conversion entre Estructuras

```
lista = ['a', 'b', 'c']
print(f"Lista: {lista}, tipo: {type(lista)}")
tupla = tuple(lista)
print(f"Tupla: {tupla}, tipo: {type(tupla)}")
conjunto = set(lista)
print(f"Set: {conjunto}, tipo: {type(conjunto)}")
```

SOLUCIONES NIVEL INTERMEDIO

Solucion I1: Agenda de Contactos

```
contactos = {}

def agregar_contacto(nombre, telefono, email):
    contactos[nombre] = {"telefono": telefono, "email": email}
    print(f"Contacto {nombre} agregado")

def buscar_contacto(nombre):
```

```
    if nombre in contactos:
        info = contactos[nombre]
        print(f"{nombre}: Tel: {info['telefono']}, Email: {info['email']}")
    else:
        print("Contacto no encontrado")

def listar_contactos():
    for nombre, info in contactos.items():
        print(f"{nombre}: {info['telefono']}")

# Uso
agregar_contacto("Ana", "555-1234", "ana@mail.com")
agregar_contacto("Luis", "555-5678", "luis@mail.com")
buscar_contacto("Ana")
listar_contactos()
```

Solucion I2: Analisis de Texto con Sets

```
texto1 = "el gato negro corre rapido"
texto2 = "el perro negro ladra fuerte"

palabras1 = set(texto1.split())
palabras2 = set(texto2.split())

print(f"Comunes: {palabras1 & palabras2}")  # {'el', 'negro'}
print(f"Solo texto1: {palabras1 - palabras2}")  # {'gato', 'corre', 'rapido'}
print(f"Total unicas: {len(palabras1 | palabras2)}")  # 8
```

Solucion I3: Lista de Tuplas Ordenada

```
estudiantes = [
    ("Ana", 95),
    ("Luis", 78),
    ("Maria", 88),
    ("Pedro", 92),
    ("Sofia", 85)
]

# Ordenar por calificacion (indice 1) de mayor a menor
estudiantes_ordenados = sorted(estudiantes, key=lambda x: x[1], reverse=True)

print("Ranking:")
for i, (nombre, calif) in enumerate(estudiantes_ordenados, 1):
    print(f"{i}. {nombre}: {calif}")
```

Solucion I4: Diccionario de Frecuencias

```
texto = "Hola Mundo Python"
frecuencias = {}

for letra in texto.lower():
    if letra != " ":  # Ignorar espacios
        if letra in frecuencias:
            frecuencias[letra] += 1
        else:
            frecuencias[letra] = 1

# Ordenar por frecuencia
```

```
for letra, freq in sorted(frecuencias.items(), key=lambda x: x[1], reverse=True):
    print(f"'{letra}': {freq}")
```

Solucion I5: Combinando Estructuras

```
# Sistema de biblioteca
libros = {
    "1984": {"autor": "George Orwell", "anio": 1949, "disponible": True},
    "Don Quijote": {"autor": "Cervantes", "anio": 1605, "disponible": True},
    "Cien anos de soledad": {"autor": "Garcia Marquez", "anio": 1967, "disponible": False}
}

generos = {"ficcion", "clasico", "realismo magico"}
prestamos = []  # Lista de tuplas (libro, usuario, fecha)

# Agregar prestamo
def prestar(titulo, usuario, fecha):
    if titulo in libros and libros[titulo]["disponible"]:
        libros[titulo]["disponible"] = False
        prestamos.append((titulo, usuario, fecha))
        print(f"{titulo} prestado a {usuario}")
    else:
        print("Libro no disponible")

prestar("1984", "Juan", "2024-01-15")
print(f"Prestamos: {prestamos}")
```

SOLUCIONES NIVEL AVANZADO

Solucion A1: Sistema de Votacion

```
candidatos = ["Ana", "Luis", "Maria"]
votos = {c: 0 for c in candidatos}  # {'Ana': 0, 'Luis': 0, 'Maria': 0}
votantes = set()  # Para evitar votos duplicados

def votar(votante_id, candidato):
    if votante_id in votantes:
        print("Error: Ya votaste")
        return
    if candidato not in candidatos:
        print("Candidato invalido")
        return
    votos[candidato] += 1
    votantes.add(votante_id)
    print(f"Voto registrado para {candidato}")

def ver_resultados():
    print("\n=== RESULTADOS ===")
    for candidato, total in sorted(votos.items(), key=lambda x: x[1], reverse=True):
        print(f"{candidato}: {total} votos")
    print(f"Total votantes: {len(votantes)}")

def declarar_ganador():
    ganador = max(votos, key=votos.get)
    print(f"\nGANADOR: {ganador} con {votos[ganador]} votos")

# Simulacion
votar("V001", "Ana")
votar("V002", "Luis")
votar("V003", "Ana")
```

```
votar("V001", "Maria")  # Error: ya voto
ver_resultados()
declarar_ganador()
```

Solucion A2: Inventario de Tienda

```
inventario = {}

def agregar_producto(codigo, nombre, precio, cantidad, categoria):
    inventario[codigo] = {
        "nombre": nombre, "precio": precio,
        "cantidad": cantidad, "categoria": categoria
    }

def vender(codigo, cantidad):
    if codigo in inventario:
        if inventario[codigo]["cantidad"] >= cantidad:
            inventario[codigo]["cantidad"] -= cantidad
            total = cantidad * inventario[codigo]["precio"]
            print(f"Vendido: {cantidad} x {inventario[codigo]['nombre']} = ${total}")
        else:
            print("Stock insuficiente")

def productos_bajo_stock(minimo=5):
    print(f"\nProductos con menos de {minimo} unidades:")
    for codigo, prod in inventario.items():
        if prod["cantidad"] < minimo:
            print(f"  {prod['nombre']}: {prod['cantidad']} unidades")

# Uso
agregar_producto("P001", "Laptop", 999, 10, "electronica")
agregar_producto("P002", "Mouse", 29, 3, "electronica")
vender("P001", 2)
productos_bajo_stock()
```

Solucion A3: Red Social Basica

```
usuarios = {}

def crear_usuario(nombre):
    usuarios[nombre] = {"amigos": set(), "publicaciones": []}

def agregar_amigo(usuario, amigo):
    if usuario in usuarios and amigo in usuarios:
        usuarios[usuario]["amigos"].add(amigo)
        usuarios[amigo]["amigos"].add(usuario)  # Amistad mutua

def publicar(usuario, texto, fecha):
    if usuario in usuarios:
        usuarios[usuario]["publicaciones"].append((texto, fecha))

def amigos_comunes(usuario1, usuario2):
    if usuario1 in usuarios and usuario2 in usuarios:
        comunes = usuarios[usuario1]["amigos"] & usuarios[usuario2]["amigos"]
        return comunes
    return set()

# Uso
```

```
crear_usuario("Ana")
crear_usuario("Luis")
crear_usuario("Maria")
agregar_amigo("Ana", "Luis")
agregar_amigo("Ana", "Maria")
agregar_amigo("Luis", "Maria")
publicar("Ana", "Hola mundo!", "2024-01-15")

print(f"Amigos de Ana: {usuarios['Ana']['amigos']}")
print(f"Amigos comunes Ana-Luis: {amigos_comunes('Ana', 'Luis')}")
```