Ejercicios de Programación

Ejercicio 1: Generar un Ticket de Compra

Descripción:
El objetivo de este ejercicio es crear un programa que genere e imprima un ticket de compra a partir de una lista de productos. Cada producto está representado por un diccionario con un nombre y un precio. El programa debe calcular la cantidad de cada producto, el subtotal de cada uno y el total de la compra.

Entrada:
Una lista de diccionarios con las claves:

- `'nombre'` (str): El nombre del producto.
- `'precio'` (float): El precio del producto.

Ejemplo de salida:

```
    -----------------
    Ticket de compra:
    -----------------
    Manzana x2 - $1.00
    Pan x1 - $1.00
    Leche x3 - $4.50
    Galletas x3 - $6.00
    -----------------
    Total: $12.50
    -----------------
```

Ejemplo de uso:

```
productos = [
    {'nombre': 'Manzana', 'precio': 0.5},
    {'nombre': 'Manzana', 'precio': 0.5},
    {'nombre': 'Pan', 'precio': 1.0},
    {'nombre': 'Leche', 'precio': 1.5}
]

generar_ticket(productos)
```

Código:

```
def generar_ticket(productos):
    contador_productos = {}
    total = 0

    for producto in productos:
        nombre = producto['nombre']
        if nombre in contador_productos:
            contador_productos[nombre]['cantidad'] += 1
        else:
            contador_productos[nombre] = {'precio': producto['precio'], 'cantidad': 1}

    print("-----------------")
    print("Ticket de compra:")
    print("-----------------")
    for nombre, info in contador_productos.items():
        precio = info['precio']
        cantidad = info['cantidad']
        subtotal = precio * cantidad
        total += subtotal
        print(f"{nombre} x{cantidad} - ${subtotal:.2f}")

    print("-----------------")
    print(f"Total: ${total:.2f}")
    print("-----------------")
```

Ejercicio 2: Seleccionar Clientes para Promoción

Descripción:
Este ejercicio se enfoca en identificar a los clientes que califican para una promoción. Los clientes califican si el monto de sus compras es mayor o igual a 150 dólares.

Entrada:
Un diccionario con:

- Claves (str): IDs de los clientes.
- Valores (float o int): Montos de las compras.

Salida:
Una lista con los IDs de los clientes que califican para la promoción.

Ejemplo de uso:

```
compras = {
    'Cliente1': 200,
    'Cliente2': 100,
```

```
    'Cliente3': 180
}

clientes_promocion = aplicar_promocion(compras)
print(clientes_promocion)  # ['Cliente1', 'Cliente3']
```

Código:

```
def aplicar_promocion(compras):
    clientes_con_promocion = []

    for cliente, monto in compras.items():
        if monto >= 150:
            clientes_con_promocion.append(cliente)

    return clientes_con_promocion
```

Otra solución:

```
def aplicar_promocion(compras):
    clientes_con_promocion = [cliente for cliente, monto in compras.items() if monto >=
150]
    return clientes_con_promocion
```

Ejercicio 3: Aplicar Descuento en Promoción

Descripción:
En este ejercicio, además de identificar a los clientes que califican para la promoción, se les aplica un descuento del 10%. El programa retorna:

1. Una lista con los IDs de los clientes que califican.
2. Un diccionario con los IDs de los clientes y sus montos ajustados después del descuento.

Entrada:
Un diccionario donde:

- Claves (str): IDs de los clientes.
- Valores (float o int): Montos de las compras.

Salida:
Dos elementos:

1. Lista de IDs de los clientes elegibles.
2. Diccionario con los montos ajustados.

Ejemplo de uso:

```
compras = {
    'Cliente1': 200,
    'Cliente2': 100,
    'Cliente3': 180
}
```

```
resultado = aplicar_promocion(compras)
print(resultado)  # [['Cliente1', 'Cliente3'], {'Cliente1': 180.0, 'Cliente3': 162.0}]
```


Código:

```
def aplicar_promocion(compras):
    clientes_con_promocion = [cliente for cliente, monto in compras.items() if monto >
150]
    cuentas_cliente_con_promocion = {cliente: monto for cliente, monto in compras.items()
if monto > 150}

    for cliente, monto in cuentas_cliente_con_promocion.items():
        cuentas_cliente_con_promocion[cliente] = round(monto * 0.9, 2)

    return [clientes_con_promocion, cuentas_cliente_con_promocion]
```

Ejercicio 4: Calcular el Costo Total de la Promoción

Descripción:
Este ejercicio calcula el costo total de la promoción, es decir, la suma de los descuentos aplicados a los clientes elegibles.

Entrada:
Un diccionario con:

- Claves (str): IDs de los clientes.
- Valores (float o int): Montos de las compras.

Salida:
Un número que representa el costo total de los descuentos aplicados.

Ejemplo de uso:

```
compras = {
    'Cliente1': 200,
    'Cliente2': 100,
    'Cliente3': 180
}

costo_promocion = calcular_costo_promocion(compras)
print(costo_promocion)  # 38.0
```

Código:

```
def aplicar_promocion(compras):
    cuentas_cliente_con_promocion = {cliente: monto for cliente, monto in compras.items()
if monto > 150}

    costo_promocion = 0

    for cliente, monto in cuentas_cliente_con_promocion.items():
        cuentas_cliente_con_promocion[cliente] = round(monto * 0.9, 2)
        costo_promocion += monto - cuentas_cliente_con_promocion[cliente]

    return [costo_promocion]
```

Ejercicio 5: Simulación de Lanzamiento de Dados

Descripción:
Vamos a mejorar un ejercicio de una sección anterior, este ejercicio simula el lanzamiento de un dado una cantidad específica de veces. El programa debe calcular la frecuencia de cada cara en porcentaje.

Entrada:
Un entero que indica cuántas veces se lanza el dado.

Salida:
Porcentaje de veces que apareció cada cara.

Ejemplo de uso:

```
tirar_dados(10)
```

Código:

```
import random

def tirar_dados(veces):
    resultados = [0] * 6

    for _ in range(veces):
        resultado = random.randint(1, 6)
        resultados[resultado - 1] += 1

    if veces == 1:
        print(f"Salió la cara: {resultado}")
    else:
        for i in range(6):
            porcentaje = (resultados[i] / veces) * 100
            print(f"Cara {i + 1}: {porcentaje:.2f}%")
```

Ejercicio 6: Filtrar y Mostrar Cursos por Estado

Descripción:
Este ejercicio organiza una lista de cursos en tres grupos según su estado:

1. En progreso.
2. Completados.
3. No iniciados.

El programa debe mostrar cada grupo con un título correspondiente.

Entrada:
Una lista de diccionarios, donde cada diccionario tiene las claves:

- `'nombre'` (str): Nombre del curso.
- `'estado'` (str): Estado del curso.

Salida:
Tres listas separadas según el estado de los cursos.

Ejemplo de uso:

```
cursos = [
        {"nombre": "HTML: Sin Fronteras", "estado": "en progreso"},
        {"nombre": "CSS3: Sin Fronteras", "estado": "completado"},
        {"nombre": "SQL: Sin Fronteras", "estado": "no iniciado"},
        {"nombre": "Python: HTML, CSS, Flask, MySQL", "estado": "en progreso"},
        {"nombre": "Aprende Javascript, HTML5, CSS3 y NodeJS desde cero", "estado":
"completado"},
        {"nombre": "React - Guía definitiva: hooks, router, redux, next + Proyectos",
```

```
"estado": "no iniciado"},
        {"nombre": "TypeScript: sin fronteras", "estado": "completado"},
        {"nombre": "Ultimate Python", "estado": "en progreso"},
        {"nombre": "Ultimate Linux", "estado": "completado"},
        {"nombre": "Ultimate Docker", "estado": "no iniciado"},
        {"nombre": "Ultimate GIT + GITHUB", "estado": "en progreso"},
        {"nombre": "Ultimate JavaScript", "estado": "completado"},
        {"nombre": "Ultimate React", "estado": "no iniciado"},
        {"nombre": "Ultimate Java", "estado": "en progreso"}
    ]

mostrar_cursos_por_estado(cursos)

salida_esperada =
    Cursos en Progreso:
     - HTML: Sin Fronteras
     - Python: HTML, CSS, Flask, MySQL
     - Ultimate Python
     - Ultimate GIT + GITHUB
     - Ultimate Java

    Cursos Completados:
     - CSS3: Sin Fronteras
     - Aprende Javascript, HTML5, CSS3 y NodeJS desde cero
     - TypeScript: sin fronteras
     - Ultimate Linux
     - Ultimate JavaScript

    Cursos No Iniciados:
     - SQL: Sin Fronteras
     - React - Guía definitiva: hooks, router, redux, next + Proyectos
     - Ultimate Docker
     - Ultimate React
```

Código:

```
def filtrar_cursos(cursos):
    # Usando filter.
    en_progreso = filter(lambda curso: curso['estado'] == 'en progreso', cursos)
    # Usando listas de comprensión.
    completados = [curso for curso in cursos if curso['estado'] == 'completado']
    no_iniciados = [curso for curso in cursos if curso['estado'] == 'no iniciado']
    return en_progreso, completados, no_iniciados

def mostrar_cursos(cursos, titulo):
    print(f"{titulo}:")
    for curso in cursos:
        print(f" - {curso['nombre']}")
    print()


def mostrar_cursos_por_estado(cursos):
    en_progreso, completados, no_iniciados = filtrar_cursos(cursos)

    mostrar_cursos(en_progreso, "Cursos en Progreso")
    mostrar_cursos(completados, "Cursos Completados")
    mostrar_cursos(no_iniciados, "Cursos No Iniciados")
```

Ejercicio 7: Calcular Mediana y Moda

Descripción:
Este ejercicio calcula dos estadísticas importantes de una lista de números:

1. La mediana: Es el valor que separa la mitad superior de la mitad inferior de una lista ordenada de números. Si la lista tiene un número par de elementos, la mediana es el promedio de los dos valores centrales.
2. La moda: El valor que aparece con mayor frecuencia.

Entrada:
Una lista de números.

Salida:
La mediana y la moda de la lista.

Ejemplo de uso:

```
datos = [1, 2, 2, 3, 4]
mediana, moda = calcular_mediana_y_moda(datos)
print(mediana, moda)  # 2, 2

datos = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 10, 10, 10]
mediana, moda = calcular_mediana_y_moda(datos)
print(mediana, moda)  # 7, 10
```

Salida esperada:

```
Mediana: 2, Moda: 2
Mediana: 7, Moda: 10
```

Código:

```
def calcular_mediana(datos):
    lista_ordenada = sorted(datos)
    n = len(lista_ordenada)
    mitad = n // 2
    if n % 2 == 0:
        return (lista_ordenada[mitad - 1] + lista_ordenada[mitad]) / 2
    else:
        return lista_ordenada[mitad]

def calcular_moda(datos):
    frecuencia = {}
    for numero in datos:
        if numero in frecuencia:
            frecuencia[numero] += 1
        else:
            frecuencia[numero] = 1
    return max(frecuencia, key=frecuencia.get)

datos = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 10, 10, 10]
mediana = calcular_mediana(datos)
moda = calcular_moda(datos)

print(f"Mediana: {mediana}, Moda: {moda}")
```

Ejercicio 8: Búsqueda en Lista de Tareas

Descripción:
Busca tareas específicas en una lista utilizando dos métodos:

1. Por ID.
2. Por texto contenido en la descripción.

Entrada:

1. Una lista de diccionarios, donde cada diccionario tiene las claves:
2. `'id'` (int): ID de la tarea.
3. `'descripcion'` (str): Descripción de la tarea.
4. Un entero para buscar por ID.
5. Un texto para buscar coincidencias.

Salida:

1. La tarea que coincide con el ID.
2. Una lista de tareas que contienen el texto.

Ejemplo de uso:

```
tareas = [
    {"id": 1, "descripcion": "Lavar los platos"},
    {"id": 2, "descripcion": "Sacar la basura"},
    {"id": 3, "descripcion": "Limpiar el baño"},
    {"id": 4, "descripcion": "Hacer la cama"},
    {"id": 5, "descripcion": "Cocinar la cena"},
    {"id": 6, "descripcion": "Pasear al perro"},
    {"id": 7, "descripcion": "Hacer la compra"},
    {"id": 8, "descripcion": "Planchar la ropa"},
    {"id": 9, "descripcion": "Regar las plantas"},
    {"id": 10, "descripcion": "Lavar el coche"}
]

buscar_tarea_por_id(tareas, 1)
buscar_tareas_por_texto(tareas, "platos")
```

Salida esperada:

```
Tarea encontrada por ID 3: {'id': 3, 'descripcion': 'Limpiar el baño'}
Tareas encontradas con el texto 'co': [{'id': 5, 'descripcion': 'Cocinar la cena'}, {'id':
7, 'descripcion': 'Hacer la compra'}, {'id': 10, 'descripcion': 'Lavar el coche'}]
Tareas encontradas con el texto 'cO': [{'id': 5, 'descripcion': 'Cocinar la cena'}, {'id':
7, 'descripcion': 'Hacer la compra'}, {'id': 10, 'descripcion': 'Lavar el coche'}]
Tareas encontradas con el texto 'almuerzo': []
```

Código:

```
def buscar_tarea_por_id(tareas, id):
    for tarea in tareas:
        if tarea["id"] == id:
            return tarea
    return None

def buscar_tareas_por_texto(tareas, texto):
    tareas_encontradas = []
    for tarea in tareas:
        if texto.lower() in tarea["descripcion"].lower():
            tareas_encontradas.append(tarea)
    return tareas_encontradas
```

Explora cada ejercicio con detalle, ejecuta el código proporcionado y realiza modificaciones para profundizar tu aprendizaje en Python.