Ejercicios de programación

Ejercicio 1: Clase Caja Registradora

Descripción:
Este ejercicio implementa una clase `CajaRegistradora` para gestionar productos y pagos en una tienda. La clase permite realizar las siguientes acciones:

- Agregar productos con su nombre y precio.
- Calcular el total de los productos agregados.
- Calcular el cambio basado en un pago proporcionado.
- Listar los productos añadidos.
- Limpiar la lista de productos después de realizar el pago.

Propiedades:

- `productos`: Lista de diccionarios que contiene los productos agregados con su nombre y precio.

Métodos:

1. `__init__()`:
   Inicializa una nueva instancia de la clase con una lista vacía de productos.
2. `agregar_producto(nombre, precio)`:
   Agrega un producto a la lista de productos especificando su nombre y precio.
3. `obtener_total()`:
   Calcula y devuelve el total de los precios de los productos en la lista.
4. `dar_cambio(pago)`:
   Calcula y devuelve el cambio basado en el pago proporcionado. Si el pago es insuficiente, lanza un mensaje indicando el problema. Limpia la lista de productos después de calcular el cambio.
5. `listar_productos()`:
   Devuelve la lista actual de productos agregados.

Pruebas:

1. **Agregar productos y obtener total:**
2. Agregar los productos `Manzana` con precio de 0.5 y `Pan` con precio de 1.0.
3. Verificar que el total sea 1.5.
4. **Calcular cambio:**
5. Al pagar con 2.0, el cambio debe ser 0.5.
6. Si se paga con 1.0, debe lanzarse una excepción indicando pago insuficiente.
7. **Listar productos:**
8. Los productos en la lista deben ser `Manzana` y `Pan`.
9. Después de pagar, la lista de productos debe quedar vacía.

Datos de prueba y salida esperada:

```
[{'nombre': 'Manzana', 'precio': 0.5}, {'nombre': 'Pan', 'precio': 1.0}]

Total: 1.5
```

```
Cambio: 0.5

[]
```

Código:

```
class CajaRegistradora:
    def __init__(self):
        self.productos = []

    def agregar_producto(self, nombre, precio):
        self.productos.append({'nombre': nombre, 'precio': precio})

    def obtener_total(self):
        return sum(producto['precio'] for producto in self.productos)

    def listar_productos(self):
        return self.productos

    def dar_cambio(self, pago):
        total = self.obtener_total()
        if pago < total:
            return("El pago es insuficiente.")
        self.productos = []
        return pago - total

caja = CajaRegistradora()

caja.agregar_producto('Manzana', 0.5)
print("Total:", caja.obtener_total())
caja.agregar_producto('Pan', 1.0)

print(caja.listar_productos())
print("Total:", caja.obtener_total())
print("Cambio:", caja.dar_cambio(2.0))
print(caja.listar_productos())
```

Ejercicio 2: Clases Caja Registradora y Cuentas

Descripción:
Este ejercicio introduce la clase `CajaRegistradora`, que permite gestionar productos y pagos, y la clase `Cuentas`, que registra múltiples cuentas, calcula el total de ventas y determina el ticket promedio.

Clase CajaRegistradora

Atributos:

- `productos`: Lista de productos agregados, cada uno representado como un diccionario con `nombre` y `precio`.

Métodos:

1. `__init__()`:
   Inicializa una lista vacía de productos.
2. `agregar_producto(nombre, precio)`:
   Agrega un producto especificando su nombre y precio.
3. `obtener_total()`:
   Calcula y devuelve el total de los precios de los productos en la lista.
4. `dar_cambio(pago)`:
   Calcula y devuelve el cambio basado en el pago proporcionado. Si el pago es insuficiente, devuelve un mensaje de error. Limpia la lista de productos tras calcular el cambio.
5. `listar_productos()`:
   Devuelve la lista actual de productos agregados.

Clase Cuentas

Atributos:

- `cuentas`: Lista de cuentas registradas, cada una con sus productos y total.

Métodos:

1. `__init__()`:
   Inicializa una lista vacía de cuentas.
2. `agregar_cuenta(caja_registradora)`:
   Agrega una cuenta a la lista basada en los productos y el total de una instancia de `CajaRegistradora`.
3. `obtener_total_cuentas()`:
   Calcula y devuelve el total de todas las cuentas registradas.
4. `obtener_ticket_promedio()`:
   Calcula y devuelve el ticket promedio de todas las cuentas. Si no hay cuentas, devuelve 0.
5. `listar_cuentas()`:
   Devuelve la lista de cuentas registradas.

Pruebas:

1. **Agregar productos y registrar cuentas:**
2. Agregar productos y registrar cuentas en `Cuentas`.
3. Verificar el total de ventas y el ticket promedio del día.
4. Listar las cuentas registradas.

Código:

```
class CajaRegistradora:
    def __init__(self):
        self.productos = []

    def agregar_producto(self, nombre, precio):
        self.productos.append({'nombre': nombre, 'precio': precio})

    def obtener_total(self):
        return sum(producto['precio'] for producto in self.productos)

    def listar_productos(self):
        return self.productos

    def dar_cambio(self, pago):
        total = self.obtener_total()
        if pago < total:
            return "El pago es insuficiente."
        self.productos = []
        return pago - total

class Cuentas:
    def __init__(self):
        self.cuentas = []

    def agregar_cuenta(self, caja_registradora):
        lista_productos = caja_registradora.listar_productos()
        total = caja_registradora.obtener_total()
        self.cuentas.append({'productos': lista_productos, 'total': total})

    def obtener_total_cuentas(self):
        return sum(cuenta['total'] for cuenta in self.cuentas)

    def obtener_ticket_promedio(self):
        if not self.cuentas:
            return 0
        return self.obtener_total_cuentas() / len(self.cuentas)

    def listar_cuentas(self):
        return self.cuentas

caja = CajaRegistradora()
registro_cuentas = Cuentas()

caja.agregar_producto('Manzana', 0.5)
print("Total:", caja.obtener_total())
caja.agregar_producto('Pan', 1.0)

print(caja.listar_productos())
```

```
print("Total:", caja.obtener_total())
registro_cuentas.agregar_cuenta(caja)
print("Cambio:", caja.dar_cambio(2.0))
print(caja.listar_productos())

caja.agregar_producto('Leche', 1.5)
print("Total:", caja.obtener_total())
registro_cuentas.agregar_cuenta(caja)
print("Cambio:", caja.dar_cambio(3.0))
print(caja.listar_productos())

caja.agregar_producto('Huevos', 2.0)
caja.agregar_producto('Queso', 3.0)
caja.agregar_producto('Jamon', 4.0)
caja.agregar_producto('Pan', 1.0)
print("Total:", caja.obtener_total())
registro_cuentas.agregar_cuenta(caja)
print("Cambio:", caja.dar_cambio(10.0))

ticket_promedio_dia = registro_cuentas.obtener_ticket_promedio()
corte_dia = registro_cuentas.obtener_total_cuentas()
cuentas = registro_cuentas.listar_cuentas()

print("Cuentas del día:", cuentas)
print(f"El ticket promedio del día es de: {ticket_promedio_dia}")
print(f"El total de ventas del día es de: {corte_dia}")
```


Ejercicio 3: Jerarquía de Clases para Vehículos

Descripción:
Este ejercicio define una jerarquía de clases para representar diferentes tipos de vehículos. Se implementa una clase base `Vehiculo` con atributos comunes y subclases especializadas para coches, motos y bicicletas.

Clases

1. **Vehiculo (Clase Base):**
   Representa un vehículo genérico con los atributos:
2. `marca`: Marca del vehículo.
3. `modelo`: Modelo del vehículo.

Métodos:

- `descripcion()`: Devuelve una descripción en formato `"Marca: <marca>, Modelo: <modelo>"`.
- **Coche (Subclase de Vehiculo):**
  Especializa la clase base para coches y redefine el método `descripcion()` para incluir el prefijo `"Coche - "`.
- **Moto (Subclase de Vehiculo):**
  Especializa la clase base para motos y redefine el método `descripcion()` para incluir el prefijo `"Moto - "`.
- **Bicicleta (Subclase de Vehiculo):**
  Especializa la clase base para bicicletas y redefine el método `descripcion()` para incluir el prefijo `"Bicicleta - "`.

Pruebas:

1. Crear instancias de cada tipo de vehículo con las siguientes marcas y modelos:
2. Coche: `"Nissan", "Versa"`
3. Moto: `"Yamaha", "MT-07"`
4. Bicicleta: `"Giant", "Escape 3"`
5. Llamar al método `descripcion()` en cada instancia y verificar la salida.

Datos de prueba y salida esperada:

```
Coche - Marca: Nissan, Modelo: Versa
Moto - Marca: Yamaha, Modelo: MT-07
Bicicleta - Marca: Giant, Modelo: Escape 3
```


Código:

```
class Vehiculo:
    def __init__(self, marca, modelo):
        self.marca = marca
        self.modelo = modelo

    def descripcion(self):
        return f"Marca: {self.marca}, Modelo: {self.modelo}"

class Coche(Vehiculo):
    def descripcion(self):
        return f"Coche - {super().descripcion()}"

class Moto(Vehiculo):
    def descripcion(self):
        return f"Moto - {super().descripcion()}"

class Bicicleta(Vehiculo):
    def descripcion(self):
        return f"Bicicleta - {super().descripcion()}"

coche = Coche("Nissan", "Versa")
moto = Moto("Yamaha", "MT-07")
bicicleta = Bicicleta("Giant", "Escape 3")

print(coche.descripcion())
print(moto.descripcion())
print(bicicleta.descripcion())
```