SOLUCIONES

Ejercicio 1

Se desea desarrollar un software para la gestión de la venta de mascarillas EPI en un establecimiento farmacéutico. Se pide:

1. Define una estructura de datos que permita almacenar información acerca de una mascarilla: se debe almacenar el modelo (un entero entre 1000 y 2000), tipo de mascarilla (FFP1, FFP2 y FFP3), color, precio y unidades disponibles. Basándote en lo anterior, crea una lista que contenga 50 modelos de tipo mascarilla (para el modelo se recomienda generar un número aleatorio) y pon su stock a 100 unidades. **[0,5 puntos]**

```
import random

mascarilla = {  'modelo': 1000,
                'tipo': 'FFP2',
                'color': 'negro',
                'precio': 0.80,
                'unidades': 150
}

lista_mascarillas = []
for i in range(50):
    mascarilla = {  'modelo': str(random.randint(1000,2000)), # string porque aunque es un
                                                              # entero no se va a operar con el
                    'tipo': random.choice(['FFP1','FFP2','FFP3']),
                    'color': random.choice(['negro','blanco','multicolor']),
                    'precio': random.uniform(0.30,0.80),
                    'unidades': 100
    }
    lista_mascarillas.append(mascarilla)
```

2. Escribe un subprograma que muestre por pantalla el stock disponible de un cierto tipo (deberá incluir todos los modelos y unidades de dicho tipo). **[1 punto]**

```
def muestra_stock_por_tipo(tipo, lista_mascarillas):
    """ int -> None
    """
    total_unidades = 0
    for mascarilla in lista_mascarillas:
        if mascarilla['tipo'] == tipo:
            print(f"Modelo: {mascarilla['modelo']}: {mascarilla['unidades']} unidades")
            total_unidades += mascarilla['unidades']
    print(f'Unidades totales de mascarillas tipo {tipo}: {total_unidades}')
```

3. Implementa un subprograma que permita realizar la venta de una mascarilla tomando como entrada el tipo de mascarilla y el número de unidades deseadas por el cliente. Si hay existencias, se mostrarán los distintos modelos de dicho tipo disponibles junto con las unidades de cada modelo y se le pedirá al usuario que elija un modelo de los disponibles. Una vez elegido, se hará efectiva la compra, que actualizará el número de unidades existentes y mostrará un mensaje indicando el importe a pagar. Si no hay existencias suficientes para satisfacer el deseo del cliente, se mostrará un mensaje indicando que no se produjo la venta. **[1,5 puntos]**

```
def stock_por_tipo(tipo, lista_mascarillas):
    """ int -> int
        OBJ: retorna el número de mascarillas en stock de un cierto tipo
    """
    total_unidades = 0
    for mascarilla in lista_mascarillas:
        if mascarilla['tipo'] == tipo:
            total_unidades += mascarilla['unidades']
    return total_unidades

def venta_mascarilla(tipo, unidades):
    """ str, int
        OBJ: Realiza una venta y actualiza stock de un tipo de mascarilla
    """
    if stock_por_tipo(tipo, lista_mascarillas) == 0:
        print('No hay suficientes existencias de ese tipo de mascarilla')
    else:
        print('Listado de modelos disponibles...')
        for mascarilla in lista_mascarillas:
            if mascarilla['tipo'] == tipo and mascarilla['unidades'] >= unidades:
                print(f"Modelo: {mascarilla['modelo']}, precio: {mascarilla['precio']}")
        modelo_elegido = input('Elija un modelo de los anteriores: ')
        i=0
        while lista_mascarillas[i]['modelo'] != modelo_elegido:
            i += 1
        lista_mascarillas[i]['unidades'] -= unidades
        precio = unidades*lista_mascarillas[i]['precio']
        print(f'Total a pagar: {precio}')
        print('La compra se ha realizado con éxito')

# prueba
venta_mascarilla('FFP2', 25)
```

4. Implementa una función que retorne una lista con todos los modelos de mascarillas que se deben reponer en la tienda. Se deben reponer todas aquellas mascarillas de las que haya menos de 100 unidades. **[1,5 puntos]**

```
def mascarillas_a_reponer(lista_mascarillas):
    """ list -> list
        OBJ: Retorna una lista con  los modelos de mascarillas que se deben reponer en tienda.
    """
    a_reponer = []
    for mascarilla in lista_mascarillas:
        if mascarilla['unidades'] < 100:
            a_reponer.append(mascarilla['modelo'])
    return a_reponer
```

5. Implementa un subprograma que permita reponer las mascarillas procedentes del distribuidor. Para ello, se tomará como entrada una lista de tuplas con modelos de mascarillas y la cantidad recibidas de cada una, con la que se actualizará el inventario de la tienda. **[1,5 puntos]**

```
def encontrar_modelo_mascarilla(modelo, lista_mascarillas):
    """ str, list -> mascarilla
        OBJ: Busca y retorna un modelo de mascarilla
    """
    for mascarilla in lista_mascarillas:
        if modelo == mascarilla['modelo']:
            return mascarilla
    return None

def reponer_mascarillas(entrada_distribuidor, lista_mascarillas):
    """ list, list -> None
        OBJ:  Permite reponer las mascarillas procedentes del distribuidor
    """
    for tupla in entrada_distribuidor:
        mascarilla = encontrar_modelo_mascarilla(tupla[0], lista_mascarillas)
        if mascarilla != None:
            mascarilla['unidades'] += tupla[1]

modelo = input('Introduce modelo a reponer: ')
reponer_mascarillas([(modelo,20)], lista_mascarillas)
```

Ejercicio 2

En una aplicación de gestión de ventas se mantiene información acerca de diversos productos, de cada uno de los cuales se almacena su código (una cadena de caracteres alfanumérica de 8 cifras, única para cada producto), nombre, precio, y existencias disponibles. El número de productos ha crecido rápidamente en los últimos años, por lo que se hace muy interesante el poder realizar búsquedas de forma más eficiente. Suponiendo que existe una función que

ordena la lista por código de manera descendente, implementa una función de búsqueda eficiente para encontrar un producto por su código. **[2 puntos]**

```
def busqueda_binaria_rec (lista_ventas, codigo, ini, fin):
    """ list, int, int, int -> pos
        OBJ: búsqueda eficiente para encontrar un producto por su código en una lista de ventas
    """
    centro = (ini+fin) // 2
    if (lista_ventas[centro] == codigo):
        pos = centro
    else :
        if (ini >fin):
            pos = None
        elif (codigo < lista_ventas[centro]):
            pos = busqueda_binaria_rec (lista_ventas, codigo, ini, centro-1)
        else :
            pos = busqueda_binaria_rec (lista_ventas, codigo, centro+1, fin)
    return pos
```

Ejercicio 3

Implementa una función recursiva que muestre los pares de valores a sumar para mostrar la suma de los extremos de una lista. Por ejemplo, para la lista [1,2,3,4,7,8,9] mostrará 1+9, en la siguiente línea 2 + 8, en la siguiente 3 + 7 y en la última 4. Si la lista tuviera un número par de valores, como [1,2,3,4] mostraría 1+4 y en la siguiente línea 2+3. **[2 puntos]**

SOLUCIÓN

```
def muestra_sumas(lista):
    """ list -> None
        OBJ:  Muestra pares de valores a sumar desde los extremos de una lista hacia el centro
    """
    longitud = len(lista)
    if longitud == 1: print(lista[0])
    elif longitud > 1:
        print(f'{lista[0]}+{lista[longitud-1]}')
        muestra_sumas(lista[1:-1])

#probador
muestra_sumas([1,2,3,4,5,6,7])
```