FUNDAMENTOS DE PROGRAMACIÓN. Curso 2019/20

PRIMER EXAMEN PRÁCTICO. 3 de febrero de 2020. Soluciones

```
# EJERCICIO 1 [1 punto]

def lee_compras(fichero):
    res = []
    with open(fichero, encoding='utf-8') as f:
        lector = csv.reader(f)
        next(lector)
        for dni, supermercado, provincia, fecha_llegada, fecha_salida,
                    total_compra in lector:
            fecha_llegada = datetime.strptime(fecha_llegada, '%d/%m/%Y %H:%M')
            fecha_salida = datetime.strptime(fecha_salida, '%d/%m/%Y %H:%M')
            total_compra = float(total_compra)
                  res.append(Compra(dni, supermercado, provincia, fecha_llegada,
                      fecha_salida, total_compra))
    return res


# EJERCICIO 2 [1 punto]

def compra_maxima_minima_provincia(compras, provincia):
    compras_provincia = [compra for compra in compras
                            if provincia is None or compra.provincia == provincia]
    maximo_compra = max(compras_provincia, key=lambda compra:compra.total_compra)
    minimo_compra = min(compras_provincia, key=lambda compra:compra.total_compra)
    return (maximo_compra.total_compra, minimo_compra.total_compra)


# EJERCICIO 3 [1,5 puntos]

def hora_menos_afluencia(compras):
    horas_llegadas = num_compras_por_hora(compras)
    return min(horas_llegadas.items(), key=lambda tupla: tupla[1])

#Función auxiliar: Creación de diccionario. Versión 1
def num_compras_por_hora(compras):
    horas_llegadas = {}
    for compra in compras:
        if compra.fecha_llegada.hour not in horas_llegadas:
            horas_llegadas[compra.fecha_llegada.hour] = 1
        else:
            horas_llegadas[compra.fecha_llegada.hour] += 1
    return horas_llegadas

#Función auxiliar: Creación de diccionario. Versión 2
def num_compras_por_hora(compras):
    horas_llegadas = Counter(compra.fecha_llegada.hour for compra in compras)
    return horas_llegadas


# EJERCICIO 4 [1,5 puntos]

def supermercados_mas_facturacion(compras, n=3):
    supermercados_facturacion = compras_por_supermercado(compras)
    ranking = sorted(supermercados_facturacion.items(),
                                key=lambda tupla:tupla[1], reverse=True)[:n]
    return list(enumerate(ranking, 1))
```

```
#Versión alternativa
def supermercados_mas_facturacion(compras, n=3):
    supermercados_facturacion = compras_por_supermercado(compras)
    lista_super = sorted(supermercados_facturacion.items(),
                         key=lambda tupla:tupla[1], reverse=True)[:n]
    return [(i+1, compra) for i, compra in enumerate(lista_super)]

#Función auxiliar: Creación de diccionario acumulador
def compras_por_supermercado(compras):
    supermercados_facturacion = {}
    for compra in compras:
        if compra.supermercado not in supermercados_facturacion:
            supermercados_facturacion[compra.supermercado] = compra.total_compra
        else:
            supermercados_facturacion[compra.supermercado] += compra.total_compra
    return supermercados_facturacion


# EJERCICIO 5 [2 puntos]

def clientes_itinerantes(compras, n):
    clientes_itinerantes = {}
    clientes_provincias = provincias_por_cliente(compras)
    for dni, conjunto_provincias in clientes_provincias.items():
        if(len(conjunto_provincias) > n):
            clientes_itinerantes[dni] = sorted(conjunto_provincias)
    return list(clientes_itinerantes.items())

#Versión por compresión
def clientes_itinerantes(compras, n):
    clientes_provincias = provincias_por_cliente(compras)
    clientes_itinerantes = {dni: sorted(conjunto_provincias)
                      for dni, conjunto_provincias in clientes_provincias.items()
                            if(len(conjunto_provincias) > n)}
    return list(clientes_itinerantes.items())

#Versión alternativa
def clientes_itinerantes(compras, n):
    clientes_itinerantes = {}
    clientes_provincias = provincias_por_cliente(compras)

    dicc = {}
    for dni in dicc_clientes.keys():
        if(len(clientes_provincias[dni]) > n):
            dicc[dni] = sorted(clientes_provincias[dni])
    return list(dicc.items())

#Función auxiliar: Creación de diccionario para agrupar por dni
def provincias_por_cliente(compras):
    clientes_provincias = {}
    for compra in compras:
        if compra.dni not in clientes_provincias:
            provincias = set()
            provincias.add(compra.provincia)
            clientes_provincias[compra.dni] = provincias
        else:
            clientes_provincias[compra.dni].add(compra.provincia)
    return clientes_provincias
```

FUNDAMENTOS DE PROGRAMACIÓN. Curso 2019/20

PRIMER EXAMEN PRÁCTICO. 3 de febrero de 2020. Soluciones

```
# EJERCICIO 6 [2 puntos]

def dias_estrella(compras, supermercado, provincia):
    dias_facturacion = compras_por_fecha_salida(compras, supermercado, provincia)
    dias_ordenados = sorted(dias_facturacion.keys())
    dias_estrella= dias(dias_ordenados, dias_facturacion)
    return dias_estrella

#Función auxiliar: días facturados
def compras_por_fecha_salida(compras, supermercado, provincia):
    dias_facturacion = {}
    for compra in compras:
        if compra.supermercado == supermercado and compra.provincia == provincia: #filtro
            if compra.fecha_salida.date() in dias_facturacion:
                dias_facturacion[compra.fecha_salida.date()] += compra.total_compra
            else:
                dias_facturacion[compra.fecha_salida.date()] = compra.total_compra
    return dias_facturacion

#Función auxiliar para el cálculo de días, con range
def dias(dias_ordenados, fact_por_dias):
    dias_estrella = list()
    for i in range(1, len(dias_ordenados) - 1):
        facturacion_ayer = fact_por_dias[dias_ordenados[i-1]]
        facturacion_hoy = fact_por_dias[dias_ordenados[i]]
        facturacion_mañana = fact_por_dias[dias_ordenados[i+1]]
        if facturacion_hoy > facturacion_ayer and facturacion_hoy > facturacion_mañana:
            dias_estrella.append(dias_ordenados[i])
    return dias_estrella

#Función auxiliar para el cálculo de días, con zip
def dias(dias_ord, fact_por_dias):
    dias_estrella = list()
    for ayer, hoy, manana in zip(dias_ord, dias_ord[1:], dias_ord[2:]):
        facturacion_ayer = fact_por_dias[ayer]
        facturacion_hoy = fact_por_dias[hoy]
        facturacion_mañana = fact_por_dias[manana]
        if facturacion_hoy > facturacion_ayer and facturacion_hoy > facturacion_mañana:
            dias_estrella.append(hoy)
    return dias_estrella


# EJERCICIO 7 [1 punto]

def test_lee_compras(compras):
    print("Número de registros leídos:", len(compras))
    print("Tres primeros registros:", compras[:3])
    print("Tres últimos registros:", compras[-2:])

def test_compra_maxima_minima_provincia(compras):
    provincia = "Huelva"
    importes = compra_maxima_minima_provincia(compras, "Huelva")
    print("Importe máximo de la provincia de", provincia, ":", importes[0],
        "Importe mínimo:", importes[1])

def test_hora_menos_afluencia(compras):
    horas = hora_menos_afluencia(compras)
    print("La hora con menos afluencia es:", horas[0], "h. con", horas[1],
        "llegadas de clientes")
```

```
def test_supermercados_mas_facturacion(compras):
    n = 2
    supermercados = supermercados_mas_facturacion(compras, n)
    print("Los", n, "Supermercados con más facturación son:", supermercados)

def test_clientes_itinerantes(compras):
    n = 7
    clientes_itinerates = clientes_itinerantes(compras, n)
    print("Los clientes itinerantes que han comprado al menos en", n, "provincias son:",
        clientes_itinerates)

def test_dias_estrella(compras):
    supermercado = "Aldi"
    provincia = "Huelva"
    dias = dias_estrella(compras, supermercado, provincia)
    print("Los días estrella del supermercado", supermercado, "de la provincia de",
        provincia, "son:", [dia_estrella.strftime('%d/%m/%Y') for dia_estrella in dias])
```