Programas que involucran datos estructurados (Diccionarios)

Ejercicio 7

Escribir un programa que cree un diccionario simulando una cesta de la compra. El programa debe preguntar el artículo y su precio y añadir el par al diccionario, hasta que el usuario decida terminar. Después se debe mostrar por pantalla la lista de la compra y el coste total, con el siguiente formato

Lista de la compra	
Artículo 1	Precio
Artículo 2	Precio
Artículo 3	Precio
…	…
Total	Coste

```
articulos = [ { "Precio": 10.5, "Articulo": "1" },
              { "Precio": 20.6, "Articulo": "2"},
              { "Precio": 9.4,  "Articulo": "3"} ]

continuar = True
while continuar:
    user_art = {
        "Articulo": raw_input("Introduce el artiuculo: "),
        "Precio": float(raw_input("Introduce el costo: "))
    }
    articulos.append(user_art)
    continuar = raw_input("Desea Continuar: ").lower() == "si"

costo = 0
print("Articulo \t Precio")
for articulo in articulos:
    print(articulo["Articulo"] + "\t\t" + str(articulo["Precio"]))
    costo += articulo["Precio"]

print("Total" + "\t\t" + str(costo))
```

Ejercicio 8

Escribir un programa que cree un diccionario de traducción español-inglés. El usuario introducirá las palabras en español e inglés separadas por dos puntos, y cada par <palabra>:<traducción> separados por comas. El programa debe crear un diccionario con las palabras y sus traducciones. Después pedirá una frase en español y utilizará el diccionario para traducirla palabra a palabra. Si una palabra no está en el diccionario debe dejarla sin traducir.

```
diccionario = { "palabra": "word", "hola":"hello", "adios": "bye" }

continuar = True
while continuar:
    palabra = raw_input("Introduce la palabra por agregar: ").lower().split(":")
    # Agrega nuevos elementos a un diccionario.
    diccionario.setdefault(palabra[0], palabra[1])
    continuar = raw_input("Desea Continuar: ").lower() == "si"

frase = raw_input("Escribe una frase: ").lower().split()

nueva_frase = ""
for palabra in frase:
    if palabra in diccionario:
        nueva_frase += diccionario[palabra] + " "
    else:
        nueva_frase += palabra + " "

print(nueva_frase.capitalize())
```

Ejercicio 9

Escribir un programa que gestione las facturas pendientes de cobro de una empresa. Las facturas se almacenarán en un diccionario donde la clave de cada factura será el número de factura y el valor el coste de la factura. El programa debe preguntar al usuario si quiere añadir una nueva factura, pagar una existente o terminar. Si desea añadir una nueva factura se preguntará por el número de factura y su coste y se añadirá al diccionario. Si se desea pagar una factura se preguntará por el número de factura y se eliminará del diccionario. Después de cada operación el programa debe mostrar por pantalla la cantidad cobrada hasta el momento y la cantidad pendiente de cobro.

```
facturas = {}

def crear_factura():
    nueva_factura = raw_input("Introduce el numero de la factura y el monto separados por una coma: ").lower().split
    facturas.setdefault(nueva_factura[0], nueva_factura[1])

def pagar_factura():
    factura = raw_input("Introduce el numero de la factura: ")
    if factura in facturas:
        facturas.pop(factura)
    else:
        print("La factura no existe.")

continuar = True
while continuar:
    accion = int(raw_input("1) Crear Factura \n2) Pagar Factura \n3)Terminar  \n Selecciona la opcion:"))

    if accion == 1:
        crear_factura()
    elif accion == 2:
        pagar_factura()
    elif accion == 3:
        continuar = False
    else:
        print("La opcion no esta disponible.")
```

Ejercicio 10

Escribir un programa que permita gestionar la base de datos de clientes de una empresa. Los clientes se guardarán en un diccionario en el que la clave de cada cliente será su NIF, y el valor será otro diccionario con los datos del cliente (nombre, dirección, teléfono, correo, preferente), donde preferente tendrá el valor True si se trata de un cliente preferente. El programa debe preguntar al usuario por una opción del siguiente menú: (1) Añadir cliente, (2) Eliminar cliente, (3) Mostrar cliente, (4) Listar todos los clientes, (5) Listar clientes preferentes, (6) Terminar. En función de la opción elegida el programa tendrá que hacer lo siguiente:

1. Preguntar los datos del cliente, crear un diccionario con los datos y añadirlo a la base de datos.
2. Preguntar por el NIF del cliente y eliminar sus datos de la base de datos.
3. Preguntar por el NIF del cliente y mostrar sus datos.
4. Mostrar lista de todos los clientes de la base datos con su NIF y nombre.
5. Mostrar la lista de clientes preferentes de la base de datos con su NIF y nombre.
6. Terminar el programa.

```python
import random

clientes = [
    {
        "nif": 258,
        "nombre": "David Iturriaga",
        "direccion": "C29A",
        "telefono": 5539565536,
        "preferente": False
    },
    {
        "nif": 256,
        "nombre": "David Medina",
        "direccion": "C60C18",
        "telefono": 58529015428,
        "preferente": True
    },
    {
        "nif": 123,
        "nombre": "Almudena Moran",
        "direccion": "casa",
        "telefono": 5561155335,
        "preferente": False
    },
    {
        "nif": 450,
        "nombre": "Ana",
        "direccion": "Satelite",
        "telefono": 5512313555,
        "preferente": True
    },
    {
        "nif": 401,
        "nombre": "Carmen Tachica",
        "direccion": "C49A",
        "telefono": 5788823541,
        "preferente": True
    }
]
```

```python
def anadir_cliente():
    cliente = {
        "nif": random.randint(1,500),
        "nombre": raw_input("Introduce tu nombre: ").capitalize(),
        "direccion": raw_input("Introduce tu direccion: "),
        "telefono": int(raw_input("Introduce tu telefono: ")),
        "correo": raw_input("Introduce tu correo: "),
        "preferente": raw_input("Introduce si eres prefrente: ").lower() == "si",
    }
    clientes.append(cliente)

def eliminar_cliente():
    nif_cliente = int(raw_input("Que cliente desea eliminar: "))
    for x in clientes:
        if x["nif"] == nif_cliente:
            clientes.remove(x)

def mostrar_cliente():
    nif_cliente = int(raw_input("Que cliente quieres revisar: "))
    for x in clientes:
        if x["nif"] == nif_cliente:
            print(x)

def mostrar_clientes():
    print(clientes)

def mostrar_clientes_preferentes():
    clientes_preferentes = []
    for x in clientes:
        if x["preferente"] == True:
            clientes_preferentes.append(x)
    print(clientes_preferentes)
```

```python
continuar = True
while continuar:
    accion = int(raw_input("1) Anadir Cliente \n2) Eliminar Cliente \n3) Mostrar Cliente \n4)Listar todos los Cliente \n5)Listar Clientes preferentes \n6) Terminar\n\n Selecciona la opcion:"

    if accion == 1:
        anadir_cliente()
    elif accion == 2:
        eliminar_cliente()
    elif accion == 3:
        mostrar_cliente()
    elif accion == 4:
        mostrar_clientes()
    elif accion == 5:
        mostrar_clientes_preferentes()
    elif accion == 6:
        continuar = False
    else:
        print("La opcion no esta disponible.")
```

Programas que involucran datos estructurados (Tuplas)

Las tuplas son similares a las listas, donde su principal diferencia es que una vez creadas no es posible editarlas.

Los **paréntesis [ ]** se utiliza para declarar una tupla.

```
tupleOfVowels = ('a', 'e', 'i', 'o', 'u')
```

Los elementos de una lista puede ser cualquier dato primitivo.

La **coma ,** se utiliza para separar los elementos dentro de una lista.

Ejercicio 11

Escribir un programa que almacene los vectores (1,2,3) y (-1,0,2) en dos listas y muestre por pantalla su producto escalar.

```
vector1 = (1,2,3)
vector2 = (-1,0,2)

total = 0
for x in range(0, len(vector1)):
    total += vector1[x] * vector2[x] # total = total + vector1[x] * vector2[x])

print("Producto escalar = " + str(total) )
```

Ejercicio 12

Escribir un programa que almacene las matrices en una lista y muestre por pantalla su producto.
Nota: Para representar matrices mediante listas usar listas anidadas, representando cada vector fila en una lista.

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \quad y \quad B = \begin{pmatrix} -1 & 0 \\ 0 & 1 \\ 1 & 1 \end{pmatrix}$$

Ejercicio 13

Escribir un programa que pregunte por una muestra de números, separados por comas, los guarde en una lista y muestre por pantalla su media y desviación típica.

```
numeros =  raw_input("Introduce una lista de numero separados por comas: ").split(",")

total_media = 0
for numero in numeros:
    total_media += int(numero)

media = total_media / len(numeros)

total_dsv = 0
for numero in numeros:
    total_dsv += (int(numero) - media) ** 2


print("media = " + str(media))
print("dv = " + str(total_dsv / len(numeros)))
```