Trabajo Práctico 4

1) Dada la lista **b = [4, "palabra", [0, 1], 9.6, False]**:
   a) Utilizar una instrucción para determinar si los siguientes elementos pertenecen a la lista:
   9.6 0 False [0, 1] 4.0 "p"
   b) Utilizar una instrucción para determinar en qué posición (índice) se encuentra el elemento [0, 1].

2) Dada la lista **L = [5, 6, [9, 5], 2, 5]**, utilizar una instrucción para contar cuántas veces aparece el elemento 5.

3) Dada la lista **a = [False, 1, 'dos', 3.0, 'ix', (5,), [3, 3], True]** utilizar instrucciones que permitan:
   a) Obtener el primero y el último elemento.
   b) Eliminar el primer elemento
   c) Eliminar los tres últimos elementos
   d) Insertar el valor 9 en el primer lugar de la lista (desplazando al resto)
   e) Reemplazar el valor 'ix' por el valor 4
   f) Agregar el valor 5 al final de la lista
   g) Obtener, mediante una rebanada, los primeros 3 elementos
   h) Insertar la lista al final de la misma lista, es decir, la lista a deberá contener a la lista a
   i) Determinar si el elemento ‘dos’ pertenece a la lista y si el elemento 3.0 pertenece a la lista
   j) Determinar cuántas veces aparece el elemento 1 en la lista y cuántas veces aparece el elemento 4
   k) Imprimir cada elemento de la lista (iterando)
   l) Imprimir cada elemento de la lista en forma inversa (iterando)

4) Escribir un programa que pida al usuario el ingreso por teclado de una serie de nombres, terminando al leer un nombre en blanco. A medida que se ingresan, agregar cada nombre al final de una lista. Al finalizar el ingreso se deben imprimir los elementos de dicha lista, iterando por ella.

5) Producir una nueva lista con los nombres de la lista generada en el ejercicio anterior, donde los nombres hayan sido convertidos completamente a mayúsculas.

6) Implementar la funcion **minimo_elemento(lista)** que recibe como parámetro una lista de elementos comparables entre sí y devuelve el mínimo valor encontrado en dicha lista o None si la lista fuera vacía.
   Ejemplos: minimo_elemento([4, 8, 15, 16, 23, 42]) retorna 4
   minimo_elemento("PYTHON") retorna 'H'

7) Escribí la función **dos_minimos(lista)** que reciba una lista de elementos comparables entre sí y devuelva los dos valores menores encontrados en la lista dada. Si la lista tuviera menos de dos elementos, retornar None por cada elemento faltante.

   Ejemplos: dos_minimos([23, 456, 12, 16, -4, 56])retorna(-4, 12)
   dos_minimos([4]) retorna (4, None)

dos_minimos([]) retorna (None, None)

8) "Piedra, papel o tijera" es un juego de manos en el cual existen tres elementos: la piedra que vence a la tijera rompiéndola, la tijera que vence al papel cortándolo y el papel que vence a la piedra envolviéndola. Escribí la función **piedra_papel_tijera(uno, dos)** que reciba dos cadenas como parámetros, donde cada una podrá ser 'piedra', 'papel' o 'tijera' (ignorando mayúsculas y minúsculas), las cuales representan los elementos elegidos por dos jugadores. La función debe retornar 1 si ganó el primer jugador, 2 si ganó el segundo o 0 si hubo empate (ambos eligieron el mismo elemento).

9) Escribí la función **digitos(numero)** que retorne una lista con los dígitos que componen al número pasado por parámetro. Ejemplo: `digitos(18413)` retornará `[1, 8, 4, 1, 3]`.

10) Escribí la función **borrar_adyacentes(lista)** que recibe una lista donde sus elementos son caracteres (strings de longitud 1) y retorna una lista en la que queda una única ocurrencia de todos los caracteres adyacentes repetidos. Ejemplo: `borrar_adyacentes(['a', 'a', '*', 'b', '=', '=', 'c', 'a'])` retornará `['a', '*', 'b', '=', 'c', 'a']`.

11) Escribí la función **ocurrencias(lista)** que recibe una lista y retorna una lista que contiene listas formadas por cada elemento de la lista junto con el número de ocurrencias contiguas de ese elemento en la lista, con el orden en que fueron apareciendo. Ejemplo: `ocurrencias(['z', 7, True, True, 34, 'z', 'z', 'z', 3.14])` retornará `[['z', 1], [7, 1], [True, 2], [34, 1], ['z', 3], [3.14, 1]]`

12) Escribí la función **sumatoria_digitos(lista)** que, dada una lista de números enteros positivos retornará una lista con la suma de los dígitos de cada uno de los números. Ejemplo: `sumatoria_digitos([154, 27890, 111, 43])` retornará `[10, 26, 3, 7]`. Para lograr su cometido, esta función deberá valerse de otra, llamada **suma_digitos(n)** que, dado un número entero n retornará la suma de sus dígitos.

13) Escribí la función **indice_mayor(lista)** que retorne el índice en el cual se encuentra el mayor número de la lista. Ejemplo: `indice_mayor([6,1,7,19,2])` retornará `3`.

14) Escribí la función **dos_sumandos(lista, resultado)** que, dada una lista compuesta por números y otro número cualquiera, retorne una nueva lista con dos elementos, que representarán índices de elementos de la lista original cuya suma da como resultado el número pasado por parámetro. De la lista original no se deberá utilizar el mismo número dos veces. Ejemplo: `dos_sumandos([2, 7, 11, 15], 17)` retornará `[0, 3]` ya que *lista[0]+lista[3] = 17*.

15) Dadas dos listas, una con datos de ciudades y otra con datos de personas, con el siguiente formato:
- La lista de ciudades contiene listas con nombres de ciudades y la provincia a la que pertenecen. Ejemplo:
[["Rosario","Santa Fe"], ["Carlos Paz","Córdoba"], ["Balcarce","Buenos Aires"], ["Cosquín","Córdoba"]]
- La lista de personas contiene listas con nombre, DNI y ciudad de cada persona. Ejemplo:

[["Juan Perez",26782345,"Carlos Paz"], ["María Gomez",40173542,"Rosario"], ["Ana Ríos",9216378,"Cosquín"]]

a) Escribir la función **obtenerCiudad(personas, DNI)** que, dada la lista de personas y el DNI de una persona, retorne la ciudad donde vive.
b) Escribir la función **obtenerProvincia(personas, ciudades, DNI)** que, dadas las dos listas y el DNI de una persona retorne la provincia donde vive. Utilizar la función del inciso a.
c) Escribir la función **contarPoblacion(personas, ciudades, provincia)** que, dadas las dos listas y una provincia, informe cuántas personas viven en esa provincia. Utilizar la función del inciso b.

16) Creá el conjunto **numeros** con los números del uno al diez. Luego, añadí el once y el doce. Actualizá la estructura incorporando los números del 30 al 35. Pensá en alguna forma de no tener que incorporarlos de a uno. Finalmente, agregá también los números 232 y -264.
a) Informar, de ser posible, el primer y el último número almacenado.
b) Informar si el 7 y el 20 pertenecen al conjunto.
c) Informar cuántos elementos hay almacenados.

17) Dados los siguientes conjuntos de hormonas alteradas en varios pacientes, se pide:
a) Informar si horm1 y horm2 comparten alguna hormona. ¿Y horm1 con horm3?
b) ¿Es el conjunto horm2 un subconjunto de horm1?
c) Informar el conjunto de todas las hormonas (iterando por el conjunto).

18) Implementá la función **digitos_repetidos(n)** que, dado un número n, retorne una lista conteniendo los dígitos que se repiten en n. Cada dígito deberá aparecer una única vez en la lista, aunque se repita varias veces.

19) Escribí un programa que solicite el ingreso de números enteros hasta leer uno que no tenga dígitos repetidos. Se pide informar:
a) Para cada número, la suma de los dígitos que se repiten en ese número.
b) Para cada número, la cantidad de dígitos que se repiten en ese número.
c) Al finalizar, el porcentaje de números procesados mayores que 478.
d) Al finalizar, el promedio de los números procesados.

20) Escribí un programa que solicite el ingreso de un string e informe:
a) La cantidad de palabras que hay en todo el texto (las palabras se separan por espacios).
b) Para cada palabra, las letras que se repiten (ignorando diferencias de mayúsculas). Sólo se deben informar letras (excluyendo dígitos y símbolos). Ejemplo: para el string “Serenos estaban los bosques…” se informará que hay 4 palabras en total y que las letras repetidas en “serenos” son la “s” y la “e”, en “estaban” es la “a”, en “bosques” es la “s”.

21) Escribí un programa que solicite el ingreso de un texto formado por una o más líneas. El ingreso finalizará cuando se lea una línea terminada con un '*'
Calcular e informar:
a) Las palabras que tienen al menos 3 vocales diferentes (ignorando diferencias de acentos y mayúsculas).

b) La cantidad de oraciones en el texto (una oración finaliza siempre con un '.').
c) El porcentaje de oraciones con más de cinco palabras.

22) Almacenar el número de habitantes de cada ciudad en una estructura que te permita acceder directamente a dicho número sabiendo el nombre de la ciudad. Los datos son: Junín: 102.023, Rojas: 28.654, Pergamino: 80.569. Escribí instrucciones que permitan:
a) Informar la cantidad de habitantes que tiene Pergamino.
b) Agregar a Lincoln con 42.036 habitantes.
c) Eliminar a Junín.
d) Incrementar los habitantes de Rojas en 1.000.
e) Modificar la cantidad de habitantes de Pergamino por 91.399.

23) Escribí un programa que almacene la información relacionada con pacientes: edad, sexo y si es diabético. Además, se registra el DNI de cada uno. Utilizá un diccionario para registrar los datos provistos en la tabla de más abajo, (deberás utilizar un contenedor para los valores). Se pide:
a) Informar cuántos pacientes hay registrados.
b) Listar todos los pacientes con su DNI y demás datos.
c) Solicitar al usuario un DNI e incrementar en 1 la edad del paciente correspondiente.
d) Indicar si los pacientes con DNI 14.972.142 y 6.409.217 sufren de diabetes.
e) Calcular la edad promedio de los pacientes con diabetes.
f) Indicar si en el diccionario existe o no algún paciente mayor de 80 años, sin diabetes (sólo decir si hay alguno o no, sin imprimir sus datos).

DNI	Edad	Sexo	Diabético
11.412.625	60	Masculino	Si
6.409.217	65	Femenino	No
19.172.162	45	Masculino	No
28.141.815	34	Femenino	Si
14.972.142	58	Masculino	Si
36.843.316	23	Femenino	No

24) Se ingresan todos los datos de los socios del club 'CAVUL'. Para cada socio se solicita: Nombre, apellido, DNI, edad y si tiene o no su cuota al día. La lectura de los datos finaliza cuando se ingresa el DNI '0'. Almacenar la información de los socios en una estructura adecuada para realizar búsquedas por DNI. Se pide:
a) Informar la cantidad de socios del club.
b) Informar la cantidad de socios morosos.
c) Informar el nombre y apellido del socio cuyo DNI es 25.123.555. En caso que no exista un socio con dicho documento, se deberá informar la situación.
d) Dar de alta a un nuevo socio cuyos datos son: DNI: 40.151.724, apellido: 'Quito', nombre: 'Esteban' y edad: 17. Dicho socio acaba de pagar su cuota inicial.
e) Informar el número de documento del socio de mayor edad.
f) Dar de baja al socio con el DNI 15.188.125, asegurándose primero que esa persona efectivamente sea socio del club. No deberá aceptarse la baja del socio si éste no tiene su cuota al día.

g) Registrar que el socio cuyo DNI es 28.731.431 acaba de pagar la cuota de este mes.

25) Escribí la función **frecuencia_caracteres(cadena)**, que recibe una cadena de texto y retorna la cantidad de veces que aparecen cada uno de los caracteres de la cadena, es decir, su frecuencia. Utilizá el tipo de datos más adecuado para retornar los datos.

26) Solicitá al usuario que ingrese nombres de ciudades y el país al que pertenecen (cortar cuando se ingresa la ciudad "zz"). Luego informá: cuántas ciudades ingresó en total y cuántas ciudades por cada país (para esto último, utilizá un diccionario). Hacé dos versiones diferentes del programa:
a) En una de ellas las ciudades no se almacenarán a medida que se las ingresa.
b) En la otra se almacenará cada ciudad con su país en una lista (cada elemento será una lista con 2 strings). Ejemplo: [["Colonia", "Uruguay"], ["Granada", "España"], ["Inverness", "Escocia"], ["Salto", "Uruguay"], ["Piriápolis", "Uruguay"], ["Aberdeen", "Escocia"]]. Luego de haber terminado la carga de datos en la lista, se procesará la misma para calcular la cantidad de ciudades por país.··

27) Se leen los datos de los productos que comercializa el supermercado 'Coto'. Para cada producto se lee: el código del producto, el tipo de producto ("lácteo", "almacén", "verdulería", "limpieza", "carnicería", "otros"), la descripción, el stock actual, el stock mínimo y el precio unitario. La lectura de datos finaliza cuando se ingresa el código de producto '-1'. Se pide almacenar los datos de los productos en una estructura adecuada para resolver de la mejor manera los siguientes puntos:
a) Determinar si es posible vender una unidad del producto con código 124.
b) Código y descripción de los productos que están por debajo del stock mínimo.
c) Cantidad de productos lácteos que comercializa el supermercado.
d) El producto de almacén que tiene menor stock actual.
e) La descripción del producto cuyo código es 3148 (puede no existir ese producto).
f) El tipo del producto con menor cantidad de unidades disponibles para la venta.
g) Un pedido puede representarse como una lista que contiene los códigos de los productos a comprar. Dado un pedido, calcular el monto total a pagar tras la compra de dichos productos y generar otra lista con los códigos de los productos que pudieron efectivamente venderse (ya que podría no haber stock de un producto que el cliente quería comprar). En el proceso, se deberá reducir el stock de cada producto efectivamente vendido.

28) Escribí un programa que permita al usuario cargar datos de una colección de películas en una estructura que permita acceso directo por título. De cada película se debe almacenar: título, director, año de estreno, listado de actores principales.
a) Informar cuántas películas dirigió cada director.
b) Informar cuál fue el director que dirigió más películas.
c) Dado el título de una película, listar todos sus actores principales.
d) Dado el nombre de un actor, listar todos los títulos de películas donde participó.

29) Escribí la función **sucesion_mira_y_deci(n)** que calcule los primeros n números de una sucesión que comienza con 1 y cada uno de los demás números se construye a partir del anterior, contando cuántas veces consecutivas aparece cada uno de sus dígitos, formando así el próximo. Por ejemplo, el siguiente de 1 es 11, ya que 1 está

formado por “un uno” => 11. El siguiente a 11 es 21 pues 11 está formado por “dos unos” => 21. El próximo a 21 es 1211 ya que 21 está formado por “un dos y un uno” => 12 11. Para más información, ver: http://es.wikipedia.org/wiki/Constante_de_Conway. Ejemplo: `sucesión_mirá_y_decí(9)` retornará `[1, 11, 21, 1211, 111221, 312211, 13112221, 1113213211, 31131211131221]`.