Universidad Católica
Silva Henríquez

Fundamentos de Programación
Ayudantía 11

Alexander Inostroza
Felipe Zambrano

Ejercicio 1

Gestor de Recetas:

Crea un programa en Python que maneje una lista de recetas. Cada receta se representará como una lista con el siguiente formato: [nombre_receta, ingrediente_1, ingrediente_2, ..., ingrediente_n].

Inicializa el programa con al menos 3 recetas predefinidas. Luego, crea un bucle que permita al usuario realizar las siguientes acciones:

Ver recetas disponibles: Muestra una lista numerada con los nombres de las recetas disponibles.

Seleccionar receta: El usuario ingresa el número correspondiente a la receta que desea visualizar.

Mostrar ingredientes: El programa muestra la lista de ingredientes de la receta seleccionada.

Salir: El bucle termina.

Ejemplo de recetas predefinidas:

["Pasta Carbonara", "Spaghetti", "Huevos", "Panceta", "Queso Pecorino", "Pimienta Negra"]

["Ensalada César", "Lechuga Romana", "Pollo a la parrilla", "Crutones", "Queso Parmesano", "Aderezo César"]

Ejercicio 2

Sistema de Gestión de Inventario de una Tienda:

Crea un programa en Python que gestione el inventario de una tienda. El inventario se representará :

[nombre_producto, categoría, precio_unitario, cantidad_en_stock]

El programa debe permitir al usuario realizar las siguientes acciones en un bucle:

Agregar producto: Solicitar al usuario la información de un nuevo producto (nombre, categoría, precio unitario, cantidad) y agregarlo al inventario.

Buscar producto por nombre: Solicitar al usuario el nombre de un producto y mostrar toda su información si se encuentra en el inventario.

Listar productos por categoría: Solicitar al usuario una categoría y mostrar la información de todos los productos que pertenecen a esa categoría.

Actualizar stock: Solicitar al usuario el nombre de un producto y la cantidad a agregar o quitar del stock. Actualizar la cantidad en stock del producto en el inventario.

Mostrar productos con stock bajo: Mostrar la información de todos los productos que tienen una cantidad en stock menor o igual a un valor definido (por ejemplo, 5 unidades).

Salir: Terminar el programa.

Ejemplo de inventario inicial:

```
inventario = [
    ["Camiseta", "Ropa", 15.99, 20],
    ["Pantalón", "Ropa", 29.99, 15],
    ["Zapatos", "Calzado", 49.99, 10],
    ["Laptop", "Electrónica", 799.99, 5],
    ["Mouse", "Electrónica", 19.99, 12]
]
```

Ejercicio 3

Escriba la función **quitar( )**, que recibirá una lista de Strings como parámetro. El contenido de la lista serán Strings con la palabra `"mantener"` o la palabra `"quitar"`. La función debe eliminar todas las apariciones de la palabra “quitar” de la lista. Suponga que dispone de RAM limitada, por lo que no puede hacer una copia de la lista original, debe modificar la lista original y **quitar( )** no retornará ningún valor.

```
>>> l = ["mantener","quitar","mantener","mantener","quitar"]
>>> quitar(l)
>>> print(l)
['mantener', 'mantener', 'mantener']
```

Ejercicio 4

Escriba la función **descomposicion_prima( )**, que recibe un número entero como argumento y retorna una lista con los factores de su descomposición prima, en cualquier orden.

```
>>> descomposicion_prima(6)
[2, 3]
>>> descomposicion_prima(36)
[2, 2, 3, 3]
>>> descomposicion_prima(13)
[13]
>>> descomposicion_prima(94)
[2, 47]
>>> descomposicion_prima(12*7*13)
[2, 2, 3, 7, 13]
```

Ejercicio 5

Como parte de un proyecto de mejoras de accesibilidad en un software, sus compañeros de la generación 2023 programaron un sistema capaz de convertir texto a voz, pero sólo es capaz de leer letras del alfabeto inglés. Para ayudarlos, programe la función **digitos( )**, que recibe un string compuesto únicamente de números y retorna una lista con la conversión de cada dígito del número a su pronunciación en español.

```
>>> digitos("123")
['uno', 'dos', 'tres']
>>> digitos("0509981")
['cero', 'cinco', 'cero', 'nueve', 'nueve', 'ocho', 'uno']
```

Ejercicio 6

Flick switch: Queremos crear una función que reciba una lista de strings como argumento y retorne una lista del mismo tamaño. Para cada elemento de la lista de entrada, la lista de salida tendrá valor de **True**, a no ser que el string sea `"flick"`; en este último caso, cambiará el valor de verdad para este elemento y todos los que continúen.

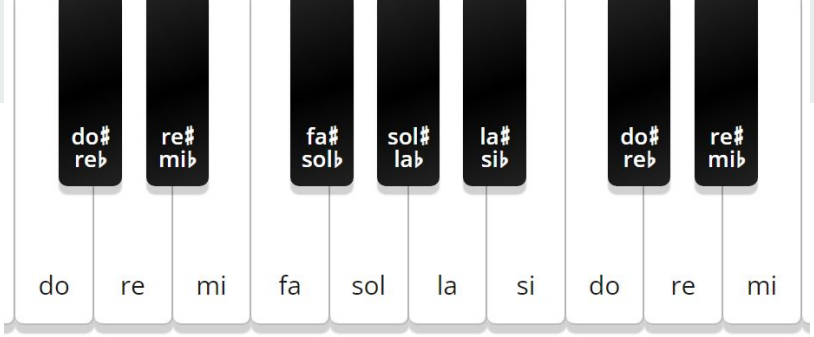
```
['codewars', 'flick', 'code', 'wars'] → [True, False, False, False]

['flick', 'chocolate', 'adventure', 'sunshine'] → [False, False, False, False]

['bicycle', 'jarmony', 'flick', 'sheep', 'flick'] → [True, True, False, False, True]
```

fuente: https://www.codewars.com/kata/64fbfe2618692c2018ebbddb

Ejercicio 7



Un amigo músico de los ayudantes quiere transponer algunas canciones para poder tocarlas en distintas tonalidades, para ello necesita nuestra ayuda. Nos explica que entre **do** y **do#** hay un semitono, y entre **do#** y **re** hay otro semitono, por lo que entre **do** y **re** hay 2 semitonos, no así entre **mi** y **fa**, que solo hay un semitono, esto se puede apreciar fácilmente en el teclado del piano. Para transponer una canción, necesita poder mover todas las notas de una melodía **n** semitonos.

Para ayudarlo, implemente la función **transponer(lista,n)**, que recibirá una lista con las notas de la melodía original y un número **n** que indica cuántos semitonos se debe transponer cada nota. Suponga que solo se usan sostenidos (#) en la notación y no bemoles.

```
>>> transponer(["mi","re#","mi","re#","mi","si","re","do","la"], -5)
['si', 'la#', 'si', 'la#', 'si', 'fa#', 'la', 'sol', 'mi']
>>> transponer(["mi","re#","mi","re#","mi","si","re","do","la"], 12)
['mi', 're#', 'mi', 're#', 'mi', 'si', 're', 'do', 'la']
```

Ejercicio 8

-

Ejercicio 9

-

Ejercicio 10

-