thePower_Proyecto3_KatasPython

Requisitos del proyecto:

Las katas pueden hacerse de distintas maneras para obtener el mismo resultado. A lo largo del proyecto debes demostrar los siguientes conocimientos:

- Manejo de tipos de datos básicos y funciones incorporadas.
- Manejo de estructuras de datos en Python y sus métodos.
- Manejo de condicionales.
- Manejo de estructuras de iteración
- Manejo de funciones en Python.
- Manejo de Clases y entendimiento de la programación orientada a objetos.
- Uso de módulos.
- Buenas prácticas.

Hay dos puntos a tener en cuenta:

1) Es importante que demuestres un entendimiento de tu código, recuerda poner comentarios explicativos de los pasos que más te hayan costado.
2) El uso de herramientas de IA está permitido pero recuerda que este proyecto está pensado para que afiances tus conocimientos en Python, asegurate de usar estás herramientas con moderación y de que entiendes todos los pasos que has seguido.

PROYECTO LÓGICA: Katas de Python

1. Escribe una función que reciba una cadena de texto como parámetro y devuelva un diccionario con las frecuencias de cada letra en la cadena. Los espacios no deben ser considerados.

```
# Pregunta 1:
# Escribe una función que reciba una cadena de texto como parámetro y devuelva un diccionario con las frecuencias de cada letra en la cadena.
# Los espacios no deben ser considerados.

def calcular_frecuencias(cadena):
    # Inicializar un diccionario vacío para almacenar las frecuencias
    frecuencias = {}

    # Iterar sobre cada carácter en la cadena
    for caracter in cadena:
        if caracter != ' ':
            if caracter in frecuencias:
                frecuencias[caracter] += 1
            else:
                frecuencias[caracter] = 1

    return frecuencias

# Ejemplo de uso
cadena = "hola mundo"
resultado = calcular_frecuencias(cadena)
print(resultado)
```

2. Dada una lista de números, obtén una nueva lista con el doble de cada valor. Usa la función map()

```
# Pregunta 2:
# Dada una lista de números, obtén una nueva lista con el doble de cada valor.
# Usa la función map().

# Ejemplo de lista de números
numeros = [1, 2, 3, 4, 5]

# Aplicamos la función map() con una función lambda para doblar cada número
dobles = list(map(lambda x: x * 2, numeros))

print(dobles)
```

3. Escribe una función que tome una lista de palabras y una palabra objetivo como parámetros. La función debe devolver una lista con todas las palabras de la lista original que contengan la palabra objetivo.

```
# Pregunta 3:
# Escribe una función que tome una lista de palabras y una palabra objetivo como parámetros.
# La función debe devolver una lista con todas las palabras de la lista original que contengan la palabra objetivo.

def palabras_con_objetivo(lista_palabras, palabra_objetivo):
    # Crear una nueva lista con las palabras que contienen la palabra objetivo
    resultado = [palabra for palabra in lista_palabras if palabra_objetivo in palabra]
    return resultado

# Ejemplo de uso
lista = ["manzana", "naranja", "plátano", "sandía", "manos"]
objetivo = "an"
resultado = palabras_con_objetivo(lista, objetivo)
print(resultado)
```

4. Genera una función que calcule la diferencia entre los valores de dos listas. Usa la función map()

```
# Pregunta 4:
# Genera una función que calcule la diferencia entre los valores de dos listas.
# Usa la función map().

def diferencias_listas(lista1, lista2):
    # Usamos map() con una función lambda para restar los elementos correspondientes de las dos listas
    diferencias = list(map(lambda x, y: x - y, lista1, lista2))
    return diferencias

# Ejemplo de uso
lista_a = [10, 20, 30]
lista_b = [5, 15, 25]
resultado = diferencias_listas(lista_a, lista_b)
print(resultado)
```

5. Ecribe una función que tome una lista de números como parámetro y un valor opcional nota_aprobado, que por defecto es 5. La función debe calcular la media de los números en la lista y determinar si la media es mayor o igual que nota aprobado. Si es así, el estado será "aprobado", de lo contrario, será "suspenso". La función debe devolver una tupla que contenga la media y el estado.

```
# Pregunta 5:
# Escribe una función que tome una lista de números y un valor opcional nota_aprobado (por defecto 5).
# La función debe calcular la media de los números y determinar si la media es mayor o igual que nota_aprobado.
# Devuelve una tupla con la media y el estado ("aprobado" o "suspenso").

def evaluar_media(numeros, nota_aprobado=5):
    # Calcular la media de la lista
    media = sum(numeros) / len(numeros) if numeros else 0

    # Determinar el estado según la media
    estado = "aprobado" if media >= nota_aprobado else "suspenso"

    # Devolver una tupla con la media y el estado
    return (media, estado)

# Ejemplo de uso
lista_numeros = [4, 6, 7, 5]
resultado = evaluar_media(lista_numeros)  # nota_aprobado por defecto 5
print(resultado)
```

6. Escribe una función que calcule el factorial de un número de manera recursiva.

```
# Pregunta 6:
# Escribe una función que calcule el factorial de un número de manera recursiva.

def factorial(n):
    if n <= 1:
        return 1
    else:
        return n * factorial(n - 1)

# Ejemplo de uso
numero = 5
resultado = factorial(numero)
print(f"El factorial de {numero} es {resultado}")
```

7. Genera una función que convierta una lista de tuplas a una lista de strings. Usa la función map()

```
# Pregunta 7:
# Genera una función que convierta una lista de tuplas a una lista de strings.
# Usa la función map().

def tuplas_a_strings(lista_tuplas):
    # Usamos map() con una función lambda para convertir cada tupla en un string
    strings = list(map(lambda t: str(t), lista_tuplas))
    return strings

# Ejemplo de uso
tuplas = [(1, 2), (3, 4), (5, 6)]
resultado = tuplas_a_strings(tuplas)
print(resultado)
```

8. Escribe un programa que pida al usuario dos números e intente dividirlos. Si el usuario ingresa un valor no numérico o intenta dividir por cero, maneja esas excepciones de manera adecuada. Asegúrate de mostrar un mensaje indicando si la división fue exitosa o no.

```
# Pregunta 8:
# Escribe un programa que pida al usuario dos números e intente dividirlos.
# Si el usuario ingresa un valor no numérico o intenta dividir por cero, maneja esas excepciones de manera adecuada.
# Muestra un mensaje indicando si la división fue exitosa o no.

try:
    num1 = float(input("Ingresa el primer número: "))
    num2 = float(input("Ingresa el segundo número: "))

    resultado = num1 / num2
    print(f"La división es exitosa. Resultado: {resultado}")
except ValueError:
    print("Error: Por favor, ingresa valores numéricos válidos.")
except ZeroDivisionError:
    print("Error: No se puede dividir por cero.")
```

9. Escribe una función que tome una lista de nombres de mascotas como parámetro y devuelva una nueva lista excluyendo ciertas mascotas prohibidas en España. La lista de mascotas a excluir es ["Mapache", "Tigre", "Serpiente Pitón", "Cocodrilo", "Oso"].Usa la función filter()

```
# Pregunta 9:
# Escribe una función que tome una lista de nombres de mascotas y devuelva una nueva lista excluyendo ciertas mascotas prohibidas en España.
# La lista de mascotas a excluir es ["Mapache", "Tigre", "Serpiente Pitón", "Cocodrilo", "Oso"]
# Usa la función filter().

def filtrar_mascotas_prohibidas(lista_mascotas):
    mascotas_prohibidas = ["Mapache", "Tigre", "Serpiente Pitón", "Cocodrilo", "Oso"]
    # Uso de filter() para excluir mascotas prohibidas
    mascotas_autorizadas = list(filter(lambda mascota: mascota not in mascotas_prohibidas, lista_mascotas))
    return mascotas_autorizadas

# Ejemplo de uso
lista = ["Perro", "Gato", "Mapache", "Loro", "Oso", "Hámster"]
resultado = filtrar_mascotas_prohibidas(lista)
print(resultado)
```

10. Escribe una función que reciba una lista de números y calcule su promedio. Si la lista está vacía, lanza una excepción personalizada y maneja el error adecuadamente.

```
# Pregunta 10:
# Escribe una función que reciba una lista de números y calcule su promedio.
# Si la lista está vacía, lanza una excepción personalizada y maneja el error adecuadamente.

# Definir una excepción personalizada
class ListaVaciaError(Exception):
    pass

def calcular_promedio(numeros):
    if not numeros:
        raise ListaVaciaError("La lista está vacía, no se puede calcular el promedio.")
    promedio = sum(numeros) / len(numeros)
    return promedio

# Ejemplo de uso con manejo de errores
try:
    lista_numeros = []
    resultado = calcular_promedio(lista_numeros)
    print(f"El promedio es: {resultado}")
except ListaVaciaError as e:
    print(f"Error: {e}")
```

11. Escribe un programa que pida al usuario que introduzca su edad. Si el usuario ingresa un valor no numérico o un valor fuera del rango esperado (por ejemplo, menor que 0 o mayor que 120), maneja las excepciones adecuadamente.

```
# Pregunta 11:
# Escribe un programa que pida al usuario que introduzca su edad.
# Si el usuario ingresa un valor no numérico o un valor fuera del rango esperado (menor que 0
o mayor que 120), maneja las excepciones adecuadamente.

try:
    edad = float(input("Por favor, ingresa tu edad: "))
    if edad < 0 or edad > 120:
        print("Error: La edad debe estar entre 0 y 120 años.")
    else:
        print(f"Tu edad es {int(edad)} años.")
except ValueError:
    print("Error: Por favor, ingresa un valor numérico válido.")
```

12. Genera una función que al recibir una frase devuelva una lista con la longitud de cada palabra. Usa la función map()

```
# Pregunta 12:
# Genera una función que al recibir una frase devuelva una lista con la longitud de cada
palabra.
# Usa la función map().

def longitudes_palabras(frase):
    # Dividir la frase en palabras
    palabras = frase.split()
    # Usar map() para obtener la longitud de cada palabra
    longitudes = list(map(len, palabras))
    return longitudes

# Ejemplo de uso
frase = "Hola, ¿cómo estás hoy?"
resultado = longitudes_palabras(frase)
print(resultado)
```

13. Genera una función la cual, para un conjunto de caracteres, devuelva una lista de tuplas con cada letra en mayúsculas y minúsculas. Las letras no pueden estar repetidas. Usa la función map()

```
# Pregunta 13:
# Genera una función la cual, para un conjunto de caracteres, devuelva una lista de tuplas con
cada letra en mayúsculas y minúsculas.
# Las letras no pueden estar repetidas. Usa la función map().

def letras_mayus_minus(conjunto):
    # Convertir el conjunto a una lista para poder mapear
    caracteres = list(conjunto)
    # Usar map() con una lambda que devuelve una tupla (mayúscula, minúscula) para cada
carácter
    resultado = list(map(lambda c: (c.upper(), c.lower()), caracteres))
    return resultado

# Ejemplo de uso
conjunto_chars = {'a', 'b', 'c'}
resultado = letras_mayus_minus(conjunto_chars)
print(resultado)
```

14. Crea una función que retorne las palabras de una lista de palabras que comience con una letra en especifico. Usa la función filter()

```
# Pregunta 14:
# Crea una función que retorne las palabras de una lista de palabras que comiencen con una letra en específico.
# Usa la función filter().

def palabras_comienzan_con(letra, lista_palabras):
    # Convertir la letra a minúscula para hacer comparación insensible a mayúsculas/minúsculas
    letra = letra.lower()
    # Usar filter() con una lambda que verifica si la palabra comienza con la letra
    resultado = list(filter(lambda palabra: palabra.lower().startswith(letra), lista_palabras))
    return resultado

# Ejemplo de uso
lista = ["Manzana", "mano", "Naranja", "Mango", "mundo"]
letra = "m"
resultado = palabras_comienzan_con(letra, lista)
print(resultado)
```

15. Crea una función lambda que sume 3 a cada número de una lista dada.

```
# Pregunta 15:
# Crea una función lambda que sume 3 a cada número de una lista dada.

# Lista de ejemplo
numeros = [1, 4, 7, 10]

sumar_tres = lambda x: x + 3

# Aplicar la lambda a cada elemento usando map()
resultado = list(map(sumar_tres, numeros))
print(resultado)
```

16. Escribe una función que tome una cadena de texto y un número entero n como parámetros y devuelva una lista de todas las palabras que sean más largas que n. Usa la función filter()

```
# Pregunta 16:
# Escribe una función que tome una cadena de texto y un número entero n como parámetrosy devuelva una lista de todas las palabras que sean más largas que n.
# Usa la función filter().

def palabras_mas_largas_que_n(cadena, n):
    palabras = cadena.split()
    # Usar filter() con una lambda para filtrar las palabras que tengan longitud mayor que n
    resultado = list(filter(lambda palabra: len(palabra) > n, palabras))
    return resultado

# Ejemplo de uso
texto = "Esto es una prueba para filtrar palabras largas"
n = 4
resultado = palabras_mas_largas_que_n(texto, n)
print(resultado)
```

17. Crea una función que tome una lista de dígitos y devuelva el número correspondiente. Por ejemplo, [5,7,2] corresponde al número quinientos setenta y dos (572). Usa la función reduce()

```
# Pregunta 17:
# Crea una función que tome una lista de dígitos y devuelva el número correspondiente.
# Por ejemplo, [5,7,2] corresponde al número 572.
# Usa la función reduce().

from functools import reduce

def lista_a_numero(digitos):
    # Usar reduce para construir el número
    numero = reduce(lambda acc, d: acc * 10 + d, digitos, 0)
    return numero

# Ejemplo de uso
digitos = [5, 7, 2]
resultado = lista_a_numero(digitos)
print(resultado)
```

18. Escribe un programa en Python que cree una lista de diccionarios que contenga información de estudiantes (nombre, edad, calificación) y use la función filter para extraer a los estudiantes con una calificación mayor o igual a 90. Usa la función filter()

```
# Pregunta 18:
# Escribe un programa en Python que cree una lista de diccionarios que contenga información de estudiantes (nombre, edad, calificación) y use la función filter para extraer a los estudiantes con una calificación mayor o igual a 90.

estudiantes = [
    {"nombre": "Ana", "edad": 20, "calificación": 95},
    {"nombre": "Luis", "edad": 22, "calificación": 88},
    {"nombre": "Carlos", "edad": 19, "calificación": 92},
    {"nombre": "María", "edad": 21, "calificación": 85},
    {"nombre": "Sofía", "edad": 23, "calificación": 97}
]

# Usar filter() para extraer estudiantes con calificación >= 90
estudiantes_destacados = list(filter(lambda e: e["calificación"] >= 90, estudiantes))

# Imprimir los estudiantes destacados
for estudiante in estudiantes_destacados:
    print(estudiante)
```

19. Crea una función lambda que filtre los números impares de una lista dada.

```
# Pregunta 19:
# Crea una función lambda que filtre los números impares de una lista dada.

# Lista de ejemplo
numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# Función lambda para filtrar impares
filtrar_impares = lambda lista: list(filter(lambda x: x % 2 != 0, lista))

# Aplicar la función lambda
impares = filtrar_impares(numeros)
print(impares)
```

20. Para una lista con elementos tipo integer y string obtén una nueva lista sólo con los valores int. Usa la función filter()

```
# Pregunta 20:
# Para una lista con elementos tipo integer y string, obtén una nueva lista solo con los valores int.
# Usa la función filter().

# Lista de ejemplo
lista_mixta = [1, "hola", 2, "mundo", 3, "python", 4]

# Usar filter() con una lambda que compruebe el tipo
solo_enteros = list(filter(lambda x: isinstance(x, int), lista_mixta))

# Mostrar resultado
print(solo_enteros)
```

21. Crea una función que calcule el cubo de un número dado mediante una función lambda

```
# Pregunta 21:
# Crea una función que calcule el cubo de un número dado mediante una función lambda.

# Función lambda para calcular el cubo
calcular_cubo = lambda x: x ** 3

# Ejemplo de uso
numero = 4
resultado = calcular_cubo(numero)
print(f"El cubo de {numero} es {resultado}")
```

22. Dada una lista numérica, obtén el producto total de los valores de dicha lista.Usa la función reduce() .

```
# Pregunta 22:
# Dada una lista numérica, obtén el producto total de los valores de dicha lista.
# Usa la función reduce().

from functools import reduce

def producto_lista(numeros):
    # Usar reduce para multiplicar todos los elementos
    producto = reduce(lambda acc, x: acc * x, numeros, 1)
    return producto

# Ejemplo de uso
lista_numeros = [2, 3, 4, 5]
resultado = producto_lista(lista_numeros)
print(f"El producto total de {lista_numeros} es {resultado}")
```

23. Concatena una lista de palabras.Usa la función reduce() .

```
# Pregunta 23:
# Concatena una lista de palabras usando la función reduce().

from functools import reduce

def concatenar_palabras(palabras):
    # Usar reduce para concatenar todas las palabras en una sola cadena, separadas por
espacio
    resultado = reduce(lambda acc, palabra: acc + ' ' + palabra if acc else palabra, palabras, "")
    return resultado

# Ejemplo de uso
lista_palabras = ["Hola", "cómo", "estás", "hoy"]
resultado = concatenar_palabras(lista_palabras)
print(resultado)
```

24. Calcula la diferencia total en los valores de una lista. Usa la función reduce() .

```
# Pregunta 24:
# Calcula la diferencia total en los valores de una lista usando la función reduce().

from functools import reduce

def diferencia_total(numeros):
    diferencia = reduce(lambda acc, x: acc - x, numeros)
    return diferencia

# Ejemplo de uso
lista_numeros = [100, 20, 30]
resultado = diferencia_total(lista_numeros)
print(f"La diferencia total en {lista_numeros} es {resultado}")
```

25. Crea una función que cuente el número de caracteres en una cadena de texto dada.

```
# Pregunta 25:
# Crea una función que cuente el número de caracteres en una cadena de texto dada.

def contar_caracteres(cadena):
    # Utilizar la función len() para contar caracteres
    return len(cadena)

# Ejemplo de uso
texto = "Hola, ¿cómo estás?"
resultado = contar_caracteres(texto)
print(f"La cantidad de caracteres en la cadena es: {resultado}")
```

26. Crea una función lambda que calcule el resto de la división entre dos números dados.

```
# Pregunta 26:
# Crea una función lambda que calcule el resto de la división entre dos números dados.

# Función lambda para calcular el resto
calcular_resto = lambda x, y: x % y

# Ejemplo de uso
num1 = 17
num2 = 5
resultado = calcular_resto(num1, num2)
print(f"El resto de {num1} dividido entre {num2} es {resultado}")
```

27. Crea una función que calcule el promedio de una lista de números.

```
# Pregunta 27:
# Crea una función que calcule el promedio de una lista de números.

def calcular_promedio(numeros):
    if not numeros:
        raise ValueError("La lista está vacía, no se puede calcular el promedio.")
    return sum(numeros) / len(numeros)

# Ejemplo de uso
lista_numeros = [10, 20, 30, 40]
resultado = calcular_promedio(lista_numeros)
print(f"El promedio de la lista es: {resultado}")
```

28. Crea una función que busque y devuelva el primer elemento duplicado en una lista dada.

```
# Pregunta 28:
# Crea una función que busque y devuelva el primer elemento duplicado en una lista dada.

def buscar_primer_duplicado(lista):
    vistos = set()
    for elemento in lista:
        if elemento in vistos:
            return elemento
        vistos.add(elemento)
    return None

# Ejemplo de uso
lista_ejemplo = [3, 5, 1, 2, 4, 5, 6]
resultado = buscar_primer_duplicado(lista_ejemplo)
print(f"El primer elemento duplicado es: {resultado}")
```

29. Crea una función que convierta una variable en una cadena de texto y enmascare todos los caracteres con el carácter '#', excepto los últimos cuatro.

```
# Pregunta 29:
# Crea una función que convierta una variable en una cadena de texto y enmascare todos los caracteres con el carácter '#', excepto los últimos cuatro.

def enmascarar_variable(variable):
    cadena = str(variable)
    if len(cadena) <= 4:
        return cadena
    else:
        return '#' * (len(cadena) - 4) + cadena[-4:]

# Ejemplo de uso
variable = 123456789
resultado = enmascarar_variable(variable)
print(f"Variable enmascarada: {resultado}")
```

30. Crea una función que determine si dos palabras son anagramas, es decir, si están formadas por las mismas letras pero en diferente orden.

```
# Pregunta 30:
# Crea una función que determine si dos palabras son anagramas, es decir, si están formadas por las mismas letras pero en diferente orden.

def son_anagramas(palabra1, palabra2):
    # Convertir las palabras a minúsculas y ordenar sus letras
    return sorted(palabra1.lower()) == sorted(palabra2.lower())

# Ejemplo de uso
palabra_a = "Escuchar"
palabra_b = "Chacurse"
resultado = son_anagramas(palabra_a, palabra_b)
print(f"Las palabras '{palabra_a}' y '{palabra_b}' son anagramas: {resultado}")
```

31. Crea una función que solicite al usuario ingresar una lista de nombres y luego solicite un nombre para buscar en esa lista. Si el nombre está en la lista, se imprime un mensaje indicando que fue encontrado, de lo contrario, se lanza una excepción.

```
# Pregunta 31:
```

```
# Crea una función que solicite al usuario ingresar una lista de nombres y luego solicite un nombre para buscar en esa lista.
# Si el nombre está en la lista, se imprime un mensaje indicando que fue encontrado, de lo contrario, se lanza una excepción.

def buscar_nombre():
    try:
        # Solicitar al usuario ingresar una lista de nombres
        nombres_entrada = input("Ingrese una lista de nombres separados por comas: ")
        lista_nombres = [nombre.strip() for nombre in nombres_entrada.split(',')]

        # Solicitar el nombre a buscar
        nombre_a_buscar = input("Ingrese el nombre a buscar: ").strip()

        # Buscar el nombre en la lista
        if nombre_a_buscar in lista_nombres:
            print(f"El nombre '{nombre_a_buscar}' fue encontrado en la lista.")
        else:
            raise ValueError(f"El nombre '{nombre_a_buscar}' no está en la lista.")
    except ValueError as e:
        print(e)

# Ejemplo de uso
buscar_nombre()
```

32. Crea una función que tome un nombre completo y una lista de empleados, busque el nombre completo en la lista y devuelve el puesto del empleado si está en la lista, de lo contrario, devuelve un mensaje indicando que la persona no trabaja aquí.

```
# Pregunta 32:
```

Crea una función que tome un nombre completo y una lista de empleados, busque el nombre completo en la lista y devuelva el puesto del empleado si está en la lista, de lo contrario, devuelve un mensaje indicando que la persona no trabaja aquí.

```
def buscar_puesto(nombre_completo, empleados):
    # empleados es una lista de diccionarios con 'nombre' y 'puesto'
    for empleado in empleados:
        if empleado['nombre'].lower() == nombre_completo.lower():
            return f"{nombre_completo} trabaja como {empleado['puesto']}."
    return f"{nombre_completo} no trabaja aquí."

# Ejemplo de uso
empleados = [
    {'nombre': 'Carlos Pérez', 'puesto': 'Gerente'},
    {'nombre': 'Ana Gómez', 'puesto': 'Analista'},
    {'nombre': 'Luis Martínez', 'puesto': 'Desarrollador'}
]

print(buscar_puesto('Ana Gomez', empleados))
print(buscar_puesto('María Lopez', empleados))
```

33. Crea una función lambda que sume elementos correspondientes de dos listas dadas.

```
# Pregunta 33:
# Crea una función lambda que sume elementos correspondientes de dos listas dadas.

# Listas de ejemplo
lista1 = [1, 2, 3]
lista2 = [4, 5, 6]

# Función lambda usando zip y map para sumar elementos correspondientes
sumar_listas = lambda l1, l2: list(map(lambda x, y: x + y, l1, l2))

# Resultado
resultado = sumar_listas(lista1, lista2)
print(resultado)
```

34. Crea la clase Arbol , define un árbol genérico con un tronco y ramas como atributos. Los métodos disponibles son: crecer_tronco , nueva_rama , crecer_ramas , quitar_rama e info_arbol . El objetivo es implementar estos métodos para manipular la estructura del árbol.

Código a seguir:

1. Inicializar un árbol con un tronco de longitud 1 y una lista vacía de ramas.

2. Implementar el método crecer_tronco para aumentar la longitud del tronco en una unidad.

3. Implementar el método nueva_rama para agregar una nueva rama de longitud 1 a la lista de ramas.

4. Implementar el método crecer_ramas para aumentar en una unidad la longitud de todas las ramas existentes.

5. Implementar el método quitar_rama para eliminar una rama en una posición específica.

6. Implementar el método info_arbol para devolver información sobre la longitud del tronco, el número de ramas y las longitudes de las mismas.

Caso de uso:

1. Crear un árbol.

2. Hacer crecer el tronco del árbol una unidad.

3. Añadir una nueva rama al árbol.

4. Hacer crecer todas las ramas del árbol una unidad.

5. Añadir dos nuevas ramas al árbol.

6. Retirar la rama situada en la posición 2.

7. Obtener información sobre el árbol.

Código Pregunta 34:

```
# Pregunta 34:
# Crear la clase Arbol con atributos y métodos para manipular su estructura.

class Arbol:
```

```
    def __init__(self):
        self.tronco = 1  # Longitud inicial del tronco
        self.ramas = []  # Lista vacía de ramas

    def crecer_tronco(self):
        """Incrementa la longitud del tronco en 1."""
        self.tronco += 1

    def nueva_rama(self):
        """Agrega una nueva rama de longitud 1 a la lista de ramas."""
        self.ramas.append(1)

    def crecer_ramas(self):
        """Incrementa en 1 la longitud de todas las ramas existentes."""
        self.ramas = [longitud + 1 for longitud in self.ramas]

    def quitar_rama(self, posicion):
        """Elimina la rama en la posición especificada (0-based)."""
        if 0 <= posicion < len(self.ramas):
            self.ramas.pop(posicion)
        else:
            print("Posición inválida para quitar rama.")

    def info_arbol(self):
        """Devuelve información del árbol: longitud del tronco, número y longitudes de las
ramas."""
        info = {
            "longitud_tronco": self.tronco,
            "numero_ramas": len(self.ramas),
            "longitudes_ramas": self.ramas
        }
        return info

# Caso de uso:
# 1. Crear un árbol
arbol = Arbol()


# 2. Hacer crecer el tronco una unidad
arbol.crecer_tronco()

# 3. Añadir una nueva rama
arbol.nueva_rama()
```

```
# 4. Hacer crecer todas las ramas una unidad
arbol.crecer_ramas()

# 5. Añadir dos nuevas ramas
arbol.nueva_rama()
arbol.nueva_rama()

# 6. Retirar la rama en la posición 2 (tercera)
arbol.quitar_rama(2)

# 7. Obtener información
info = arbol.info_arbol()
print(info)
```

35. Crea la clase UsuarioBanco ,representa a un usuario de un banco con su nombre, saldo y si tiene o no cuenta corriente. Proporciona métodos para realizar operaciones como retirar dinero, transferir dinero desde otro usuario y agregar dinero al saldo.

Código a seguir:

1. Inicializar un usuario con su nombre, saldo y si tiene o no cuenta corriente mediante True y False .

2. Implementar el método retirar_dinero para retirar dinero del saldo del usuario. Lanzará un error en caso de no poder hacerse.

3. Implementar el método transferir_dinero para realizar una transferencia desde otro usuario al usuario actual. Lanzará un error en caso de no poder hacerse.

4. Implementar el método agregar_dinero para agregar dinero al saldo del usuario.

Caso de uso:

1. Crear dos usuarios: "Alicia" con saldo inicial de 100 y "Bob" con saldo inicial de 50, ambos con cuenta corriente.

2. Agregar 20 unidades de saldo de "Bob".

3. Hacer una transferencia de 80 unidades desde "Bob" a "Alicia".

4. Retirar 50 unidades de saldo a "Alicia".

Código Pregunta 35:

```
# Pregunta 35:
# Crear la clase UsuarioBanco con operaciones de saldo y transferencias.

class UsuarioBanco:
    def __init__(self, nombre, saldo_inicial, tiene_cuenta_corriente):
        self.nombre = nombre
        self.saldo = saldo_inicial
        self.tiene_cuenta_corriente = tiene_cuenta_corriente

    def retirar_dinero(self, cantidad):
        if cantidad > self.saldo:
            raise ValueError(f"No hay suficiente saldo para retirar {cantidad}. Saldo actual:
{self.saldo}")
        self.saldo -= cantidad

    def transferir_dinero(self, otro_usuario, cantidad):
        if not isinstance(otro_usuario, UsuarioBanco):
            raise TypeError("El destinatario debe ser una instancia de UsuarioBanco.")
        if cantidad > self.saldo:
            raise ValueError(f"No hay suficiente saldo para transferir {cantidad}. Saldo actual:
{self.saldo}")
```

```
        # Transferencia
        self.retirar_dinero(cantidad)
        otro_usuario.agregar_dinero(cantidad)

    def agregar_dinero(self, cantidad):
        self.saldo += cantidad

# Caso de uso:
# 1. Crear dos usuarios: "Alicia" con saldo 100, "Bob" con saldo 50, ambos con cuenta corriente
alice = UsuarioBanco("Alicia", 100, True)
bob = UsuarioBanco("Bob", 50, True)

# 2. Agregar 20 unidades a "Bob"
bob.agregar_dinero(20)

# 3. Transferir 80 unidades desde "Bob" a "Alicia"
try:
    bob.transferir_dinero(alice, 80)
except Exception as e:
    print(e)

# 4. Retirar 50 unidades de saldo a "Alicia"
try:
    alice.retirar_dinero(50)
except Exception as e:
    print(e)

# Mostrar saldos finales
print(f"{alice.nombre} saldo: {alice.saldo}")
print(f"{bob.nombre} saldo: {bob.saldo}")
```

36. Crea una función llamada procesar_texto que procesa un texto según la opción especificada: contar_palabras , reemplazar_palabras , eliminar_palabra . Estas opciones son otras funciones que tenemos que definir primero y llamar dentro de la función procesar_texto .

Código a seguir:

1. Crear una función contar_palabras para contar el número de veces que aparece cada palabra en el texto. Tieneque devolver un diccionario.

2. Crear una función reemplazar_palabras para remplazar una palabra_original del texto por una palabra_nueva . Tiene que devolver el texto con el remplazo de palabras.

3. Crear una función eliminar_palabra para eliminar una palabra del texto. Tiene que devolver el texto con la palabra eliminada.

4. Crear la función procesar_texto que tome un texto, una opción(entre "contar", "reemplazar", "eliminar") y un número de argumentos variable según la opción indicada.

Caso de uso:

Comprueba el funcionamiento completo de la función procesar_texto

Código Pregunta 36:

```
# Pregunta 36:
# Funciones para procesar un texto según diferentes opciones.

def contar_palabras(texto):
    """Cuenta la cantidad de veces que aparece cada palabra en el texto."""
    palabras = texto.lower().split()
    conteo = {}
    for palabra in palabras:
        conteo[palabra] = conteo.get(palabra, 0) + 1
    return conteo

def reemplazar_palabras(texto, palabra_original, palabra_nueva):
    """Reemplaza todas las ocurrencias de palabra_original por palabra_nueva en el texto."""
    palabras = texto.split()
    palabras_reemplazadas = [palabra_nueva if p == palabra_original else p for p in palabras]
    return ' '.join(palabras_reemplazadas)

def eliminar_palabra(texto, palabra_eliminar):
    """Elimina todas las ocurrencias de palabra_eliminar del texto."""
    palabras = texto.split()
    palabras_filtradas = [p for p in palabras if p != palabra_eliminar]
    return ' '.join(palabras_filtradas)

def procesar_texto(texto, opcion, *args):
    """Procesa el texto según la opción especificada."""
```

```
    if opcion == "contar":
        return contar_palabras(texto)
    elif opcion == "reemplazar":
        if len(args) != 2:
            raise ValueError("Reemplazar requiere dos argumentos: palabra_original y
palabra_nueva.")
        return reemplazar_palabras(texto, args[0], args[1])
    elif opcion == "eliminar":
        if len(args) != 1:
            raise ValueError("Eliminar requiere un argumento: palabra_eliminar.")
        return eliminar_palabra(texto, args[0])
    else:
        raise ValueError("Opción inválida. Use 'contar', 'reemplazar' o 'eliminar'.")

# Caso de uso completo
texto_ejemplo = "Hola mundo. Este es un ejemplo de procesamiento de texto. Mundo, mundo!"
print("Contar palabras:", procesar_texto(texto_ejemplo, "contar"))
print("Reemplazar 'mundo' por 'universo':", procesar_texto(texto_ejemplo, "reemplazar",
"mundo", "universo"))
print("Eliminar 'mundo':", procesar_texto(texto_ejemplo, "eliminar", "mundo"))
```

37. Genera un programa que nos diga si es de noche, de día o tarde según la hora proporcionada por el usuario.

```
# Pregunta 37:
# Programa que indica si es de noche, día o tarde según la hora proporcionada.

def determinar_tiempo(hora):
    """Devuelve 'noche', 'día' o 'tarde' según la hora en 24 horas."""
    if not (0 <= hora <= 23):
```

```
        raise ValueError("La hora debe estar entre 0 y 23.")
    if 6 <= hora < 12:
        return "Es día."
    elif 12 <= hora < 20:
        return "Es tarde."
    else:
        return "Es noche."

try:
    hora_usuario = int(input("Ingresa la hora en formato 24 horas (0-23): "))
    resultado = determinar_tiempo(hora_usuario)
    print(resultado)
except ValueError as e:
    print(f"Error: {e}")
```

38. Escribe un programa que determine qué calificación en texto tiene un alumno en base a su calificación numérica.

Las reglas de calificación son:

- **0 - 69 insuficiente**
- **70 - 79 bien**
- **80 - 89 muy bien**

- 90 - 100 excelente

Código Pregunta 38:

```
# Pregunta 38:
# Programa que determina la calificación en texto basada en la calificación numérica.

def determinar_calificacion(nota):
    """Devuelve la calificación en texto según la nota numérica."""
    if not (0 <= nota <= 100):
        raise ValueError("La nota debe estar entre 0 y 100.")
    if 0 <= nota <= 69:
        return "Insuficiente"
    elif 70 <= nota <= 79:
        return "Bien"
    elif 80 <= nota <= 89:
        return "Muy bien"
    elif 90 <= nota <= 100:
        return "Excelente"

try:
    nota_usuario = float(input("Ingresa la calificación numérica (0-100): "))
    calificacion_texto = determinar_calificacion(nota_usuario)
    print(f"La calificación en texto es: {calificacion_texto}")
except ValueError as e:
    print(f"Error: {e}")
```

39. Escribe una función que tome dos parámetros: figura (una cadena que puede ser "rectangulo" , "circulo" o "triangulo") y datos (una tupla con los datos necesarios para calcular el área de la figura).

```
# Pregunta 39:
# Función que calcula el área de una figura según su tipo y datos.

def calcular_area(figura, datos):
    if figura == "rectangulo":
        # Datos: (base, altura)
        base, altura = datos
```

```
        return base * altura
    elif figura == "circulo":
        # Datos: (radio,)
        radio, = datos
        from math import pi
        return pi * radio ** 2
    elif figura == "triangulo":
        # Datos: (base, altura)
        base, altura = datos
        return (base * altura) / 2
    else:
        raise ValueError("Figura no reconocida. Use 'rectangulo', 'circulo' o 'triangulo'.")

# Ejemplo de uso:
area_rectangulo = calcular_area("rectangulo", (5, 10))
area_circulo = calcular_area("circulo", (7,))
area_triangulo = calcular_area("triangulo", (6, 8))

print(f"Área de rectángulo: {area_rectangulo}")
print(f"Área de círculo: {area_circulo}")
print(f"Área de triángulo: {area_triangulo}")
```

40. En este ejercicio, se te pedirá que escribas un programa en Python que utilice condicionales para determinar el monto final de una compra en una tienda en línea, después de aplicar un descuento. El programa debe hacer lo siguiente:

1. Solicita al usuario que ingrese el precio original de un artículo.

2. Pregunta al usuario si tiene un cupón de descuento (respuesta sí o no).

3. Si el usuario responde que sí, solicita que ingrese el valor del cupón de descuento.

4. Aplica el descuento al precio original del artículo, siempre y cuando el valor del cupón sea válido (es decir, mayor a cero). Por ejemplo, descuento de 15€.

5. Muestra el precio final de la compra, teniendo en cuenta el descuento aplicado o sin él.

6. Recuerda utilizar estructuras de control de flujo como if, elif y else para llevar a cabo estas acciones en tu programa de Python.

Código Pregunta 40:

```
# Pregunta 40:
# Programa que calcula el monto final de una compra con descuento en una tienda en línea.

# Paso 1: Solicitar precio original
precio_original = float(input("Ingresa el precio original del artículo: € "))

# Paso 2: Preguntar si tiene cupón de descuento
respuesta = input("¿Tienes un cupón de descuento? (sí/no): ").strip().lower()

descuento = 0  # Valor por defecto del descuento

# Paso 3 y 4: Validar respuesta y aplicar descuento si procede
if respuesta == "sí":
    valor_cupon = float(input("Ingresa el valor del cupón de descuento en €: "))
    if valor_cupon > 0:
        descuento = valor_cupon
    else:
        print("Valor del cupón inválido. No se aplicará descuento.")
elif respuesta == "no":
    print("No se aplicará ningún descuento.")
else:
    print("Respuesta no válida. Asumiendo que no hay descuento.")

# Paso 5: Calcular precio final
precio_final = precio_original - descuento
# Asegurarnos de que el precio final no sea negativo
precio_final = max(precio_final, 0)

# Paso 6: Mostrar resultado
print(f"El precio final de la compra es: € {precio_final:.2f}")
```