MAESTRÍA EN INVESTIGACIÓN EN CIENCIA DE DATOS
FACULTAD DE CIENCIAS FÍSICO MATEMÁTICAS

GUÍA DE EXAMEN: Programación en Python

Este documento presenta ejercicios de ejemplo como preparación para el examen de admisión a la maestría en investigación en ciencia de datos.

IMPORTANTE

- El examen de admisión se realizará "offline" en las computadoras del laboratorio de cómputo de la Facultad de Ciencias Físico Matemáticas (BUAP). El día del examen, todos los documentos necesarios estarán disponibles en carpeta local.
- Esta sección del examen de admisión tendrá como aspectos a evaluar la funcionalidad de código (70%) y la calidad del código realizado (30%).

Parte 01

> 1.- Define una función en Python llamada `buscar_valor` que tome dos argumentos: una lista de números `nums` y un valor `buscar`. La función debe devolver una lista de índices donde se encuentra el valor `buscar` en la lista `nums`.

In [2]:
```
# Respuesta:
def buscar_valor(nums, buscar):
    return [i for i, num in enumerate(nums) if num == buscar]

nums = [1, 2, 3, 4, 2, 5, 2]
buscar = 2
print("Los índices donde se encuentra", buscar, "son:", buscar_valor(n
```

```
Los índices donde se encuentra 2 son: [1, 4, 6]
```

2.- Define una clase llamada `Pila` que implemente una pila utilizando una lista. La clase debe tener métodos para agregar un elemento (`push`), eliminar un elemento (`pop`) y verificar si la pila está vacía (`esta_vacia`).

In [3]:
```python
# Respuesta:
class Pila:
    def __init__(self):
        self.items = []

    def push(self, item):
        self.items.append(item)

    def pop(self):
        if not self.esta_vacia():
            return self.items.pop()
        else:
            raise IndexError("La pila está vacía")

    def esta_vacia(self):
        return len(self.items) == 0

# Ejemplo de uso:
pila = Pila()
pila.push(1)
pila.push(2)
print("Elemento retirado de la pila:", pila.pop())
print("¿La pila está vacía?", pila.esta_vacia())
```

```
Elemento retirado de la pila: 2
¿La pila está vacía? False
```

Escribe una función en Python llamada `palindromo` que tome una cadena como argumento y devuelva `True` si la cadena es un palíndromo (se lee igual hacia adelante que hacia atrás) y `False` de lo contrario.

In [4]:
```
# Respuesta:
def palindromo(cadena):
    return cadena == cadena[::-1]

cadena = "anitalavalatina"
print("¿Es un palíndromo?", palindromo(cadena))
```

```
¿Es un palíndromo? True
```

3.- Define una clase llamada `Matriz` que represente una matriz utilizando una lista de listas en Python. La clase debe tener métodos para sumar dos matrices (`sumar`), multiplicar dos matrices (`multiplicar`), y encontrar la transpuesta de la matriz (`transpuesta`).

In [11]:
```
import numpy as np

class Matriz:
    def __init__(self, matriz):
        self.matriz = np.array(matriz)

    def sumar(self, otra_matriz):
        if self.matriz.shape != otra_matriz.matriz.shape:
            raise ValueError("Las matrices deben tener la misma forma
        return Matriz(self.matriz + otra_matriz.matriz)

    def multiplicar(self, otra_matriz):
        if self.matriz.shape[1] != otra_matriz.matriz.shape[0]:
            raise ValueError("El número de columnas de la primera matr
        return Matriz(np.dot(self.matriz, otra_matriz.matriz))

    def transpuesta(self):
        return Matriz(self.matriz.transpose())

# Ejemplo de uso:
matriz_a = Matriz([[1, 2], [3, 4]])
matriz_b = Matriz([[5, 6], [7, 8]])

print("Matriz A:")
print(matriz_a.matriz)
print("Matriz B:")
print(matriz_b.matriz)

print("Suma de matrices:")
print(matriz_a.sumar(matriz_b).matriz)
```

```
print("Producto de matrices:")
print(matriz_a.multiplicar(matriz_b).matriz)

print("Transpuesta de matriz A:")
print(matriz_a.transpuesta().matriz)
```

```
Matriz A:
[[1 2]
 [3 4]]
Matriz B:
[[5 6]
 [7 8]]
Suma de matrices:
[[ 6  8]
 [10 12]]
Producto de matrices:
[[19 22]
 [43 50]]
Transpuesta de matriz A:
[[1 3]
 [2 4]]
```

4.- Escribe un decorador en Python llamado `cronometrar` que pueda calcular el tiempo de ejecución de cualquier función y lo imprima.

In [12]:
```
# Respuesta:
import time

def cronometrar(funcion):
    def wrapper(*args, **kwargs):
        inicio = time.time()
        resultado = funcion(*args, **kwargs)
        fin = time.time()
        print(f"Tiempo de ejecución de {funcion.__name__}: {fin - inic
        return resultado
    return wrapper

# Ejemplo de uso:
@cronometrar
def ejemplo():
    time.sleep(2)
    return "Hecho"

ejemplo()
```

```
Tiempo de ejecución de ejemplo: 2.0040712356567383 segundos
```

Out[12]:
```
'Hecho'
```

> 5.- Define una función generadora en Python llamada `generar_fibonacci` que genere la secuencia de Fibonacci hasta el n-ésimo término.

In [13]:
```
# Respuesta:
def generar_fibonacci(n):
    a, b = 0, 1
    for _ in range(n):
        yield a
        a, b = b, a + b

# Ejemplo de uso:
n = 10
fibonacci = generar_fibonacci(n)
print("Secuencia de Fibonacci hasta el término", n, ":", list(fibonacc
```

```
Secuencia de Fibonacci hasta el término 10 : [0, 1, 1, 2, 3, 5, 8, 1
3, 21, 34]
```

6.- Escribe una función en Python llamada `leer_archivo` que tome el nombre de un archivo como argumento y devuelva el contenido del archivo en forma de lista de líneas, pero sin incluir líneas vacías ni líneas que comiencen con "#" (comentarios).

In [18]:
```
# Respuesta:
def leer_archivo(nombre_archivo):
    with open(nombre_archivo, 'r') as archivo:
        return [linea.strip() for linea in archivo.readlines() if line

# Ejemplo de uso:
nombre_archivo = "archivo.txt"
contenido = leer_archivo(nombre_archivo)
print("Contenido del archivo sin comentarios ni líneas vacías:")
for linea in contenido:
    print(linea)
```

```
Contenido del archivo sin comentarios ni líneas vacías:
Hola bienvenidos "Hijos de Newton "
Hasta luego
```

7.- Escribe una función en Python llamada `calcular_promedio` que tome una lista de números y calcule el promedio de esos números, pero excluyendo los valores más bajos y más altos.

In [19]:
```
# Respuesta:
def calcular_promedio(lista):
    lista_ordenada = sorted(lista)[1:-1]
    return sum(lista_ordenada) / len(lista_ordenada)

numeros = [1, 2, 3, 4, 5]
print("Promedio excluyendo valores más bajos y más altos:", calcular_p
```

```
Promedio excluyendo valores más bajos y más altos: 3.0
```

8.- Escribe una función en Python llamada `combinaciones_unicas` que tome una lista de números y devuelva una lista de todas las combinaciones únicas de longitud `k` .

In [24]:
```
# Respuesta:
from itertools import combinations

def combinaciones_unicas(lista, k):
    return list(combinations(lista, k))

numeros = [1, 2, 3, 4]

# ejemplo:
print("Combinaciones únicas de longitud 3:", combinaciones_unicas(nume
```

```
Combinaciones únicas de longitud 3: [(1, 2, 3), (1, 2, 4), (1, 3, 4),
(2, 3, 4)]
```

9.- Escribe una función en Python llamada `interseccion_diccionarios` que tome dos diccionarios como argumentos y devuelva un nuevo diccionario que contenga las claves que están presentes en ambos diccionarios y cuyos valores sean iguales.

In [26]:
```
# Respuesta:
def interseccion_diccionarios(dict1, dict2):
    return {k: dict1[k] for k in dict1 if k in dict2 and dict1[k] == d

diccionario1 = {"a": 1, "b": 2, "c": 3}
diccionario2 = {"b": 2, "c": 3, "d": 4}
print("Intersección de diccionarios:", interseccion_diccionarios(dicci
```

```
Intersección de diccionarios: {'b': 2, 'c': 3}
```

10.- Escribe una función en Python llamada `encontrar_primos` que tome un número entero positivo `n` como argumento y devuelva una lista de todos los números primos menores o iguales a `n`.

In [27]:
```
# Respuesta:
def encontrar_primos(n):
    primos = []
    for num in range(2, n+1):
        es_primo = True
        for i in range(2, int(num**0.5) + 1):
            if num % i == 0:
                es_primo = False
                break
        if es_primo:
            primos.append(num)
    return primos

n = 20
print("Números primos menores o iguales a", n, ":", encontrar_primos(n
```

```
Números primos menores o iguales a 20 : [2, 3, 5, 7, 11, 13, 17, 19]
```

11.-Escribe una función en Python llamada `mezclar_listas` que tome dos listas como argumentos y devuelva una nueva lista que contenga los elementos de ambas listas, alternando entre ellas.

In [1]:
```
# Respuesta:
def mezclar_listas(lista1, lista2):
    mezclada = []
    for elemento1, elemento2 in zip(lista1, lista2):
        mezclada.append(elemento1)
        mezclada.append(elemento2)
    return mezclada

lista1 = [1, 3, 5]
lista2 = [2, 4, 6]
print("Lista mezclada:", mezclar_listas(lista1, lista2))
```

```
Lista mezclada: [1, 2, 3, 4, 5, 6]
```

Parte 02

> 1.- Escribe una función utilizando NumPy que tome una matriz y devuelva la matriz transpuesta multiplicada por ella misma.

In [29]:
```
import numpy as np

def matriz_transpuesta_multiplicada(matriz):
    transpuesta = matriz.T
    return np.dot(transpuesta, matriz)

# Ejemplo de uso:
matriz = np.array([[1, 2],
                   [3, 4]])
resultado = matriz_transpuesta_multiplicada(matriz)
print("Matriz transpuesta multiplicada por sí misma:")
print(resultado)
```

```
Matriz transpuesta multiplicada por sí misma:
[[10 14]
 [14 20]]
```

> 2.- Utilizando Pandas , carga un archivo CSV que contiene datos de ventas con columnas de fecha y cantidad vendida. Luego, calcula el total de ventas por mes y traza un gráfico de barras mostrando esta información.

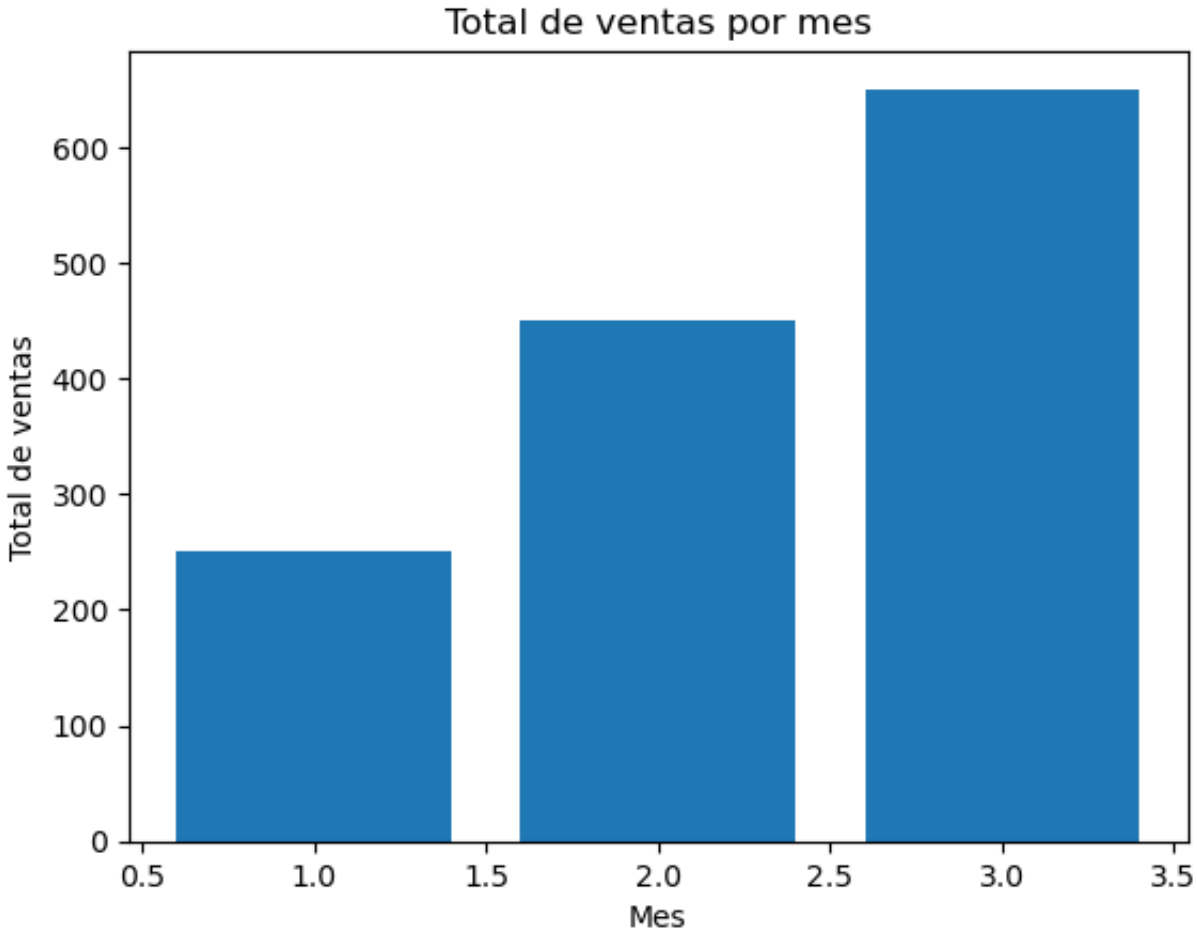
In [33]:
```
!cat ventas.csv
```

```
Fecha,Cantidad_Vendida
2022-01-01,100
2022-01-05,150
2022-02-10,200
2022-02-15,250
2022-03-20,300
2022-03-25,350
```

In [32]:

In [32]:

```
import pandas as pd
import matplotlib.pyplot as plt

# Cargar archivo CSV
datos_ventas = pd.read_csv('ventas.csv')

# Convertir la columna de fecha a tipo datetime
datos_ventas['Fecha'] = pd.to_datetime(datos_ventas['Fecha'])

# Agregar una columna de mes
datos_ventas['Mes'] = datos_ventas['Fecha'].dt.month

# Calcular el total de ventas por mes
ventas_por_mes = datos_ventas.groupby('Mes')['Cantidad_Vendida'].sum()

# Gráfico de barras
plt.bar(ventas_por_mes.index, ventas_por_mes.values)
plt.xlabel('Mes')
plt.ylabel('Total de ventas')
plt.title('Total de ventas por mes')
plt.show()
```



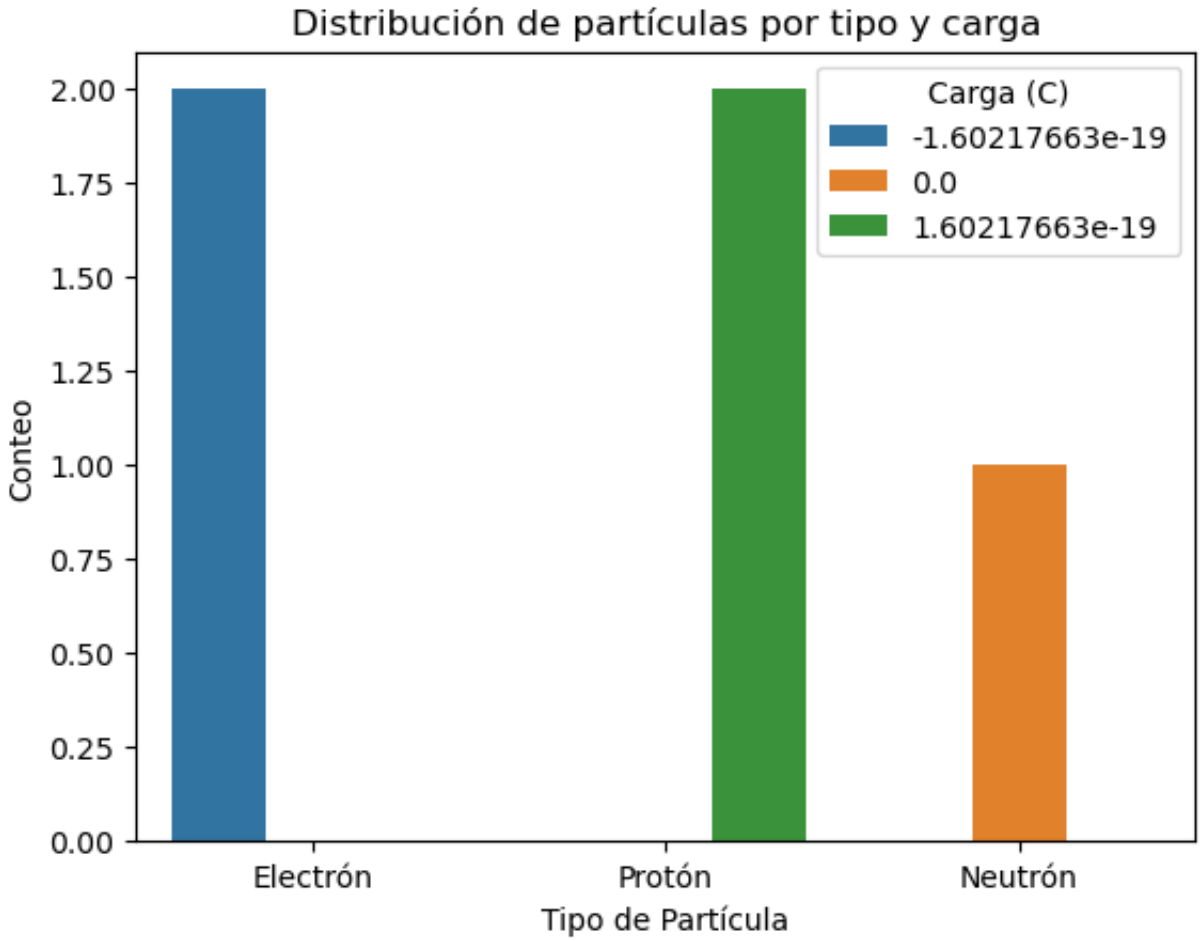
3.- Utiliza `Pandas` , `Matplotlib` y `Seaborn` para trazar para trazar un gráfico de barras apiladas que muestre la distribución del tipo de partícula con respecto a la carga en un `DataFrame` , primero crea un `DataFrame` llamado `datos que contiene información sobre partículas, donde cada fila representa una partícula y las columnas contienen información sobre diferentes características de las partículas, como tipo de partícula, masa y carga.

In [43]:
```
!cat datos.csv
```

```
Tipo de Partícula,Masa (kg),Carga (C)
Electrón,9.10938356e-31,-1.60217663e-19
Protón,1.6726219e-27,1.60217663e-19
Neutrón,1.6749275e-27,0.0
Electrón,9.10938356e-31,-1.60217663e-19
Protón,1.6726219e-27,1.60217663e-19
```

In [39]:
```
import pandas as pd

# Crear un DataFrame de ejemplo
datos = pd.DataFrame({
    'Tipo de Partícula': ['Electrón', 'Protón', 'Neutrón', 'Electrón',
    'Masa (kg)': [9.10938356e-31, 1.6726219e-27, 1.6749275e-27, 9.1093
    'Carga (C)': [-1.60217663e-19, 1.60217663e-19, 0, -1.60217663e-19,
})


# Guardar el DataFrame en un archivo CSV
datos.to_csv('datos.csv', index=False)
```

In [40]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Gráfico de barras apiladas
sns.countplot(x='Tipo de Partícula', hue='Carga (C)', data=datos)
plt.xlabel('Tipo de Partícula')
plt.ylabel('Conteo')
plt.title('Distribución de partículas por tipo y carga')
plt.legend(title='Carga (C)')
plt.show()
```



4.- Escribe una función utilizando NumPy que genere una matriz de tamaño `n x n` donde cada elemento `i, j` es la suma de los valores `i` y `j` de dos matrices de tamaño `n x n` dadas.

In [45]:
```
import numpy as np

def suma_matrices_elemento_a_elemento(matriz1, matriz2):
    return matriz1 + matriz2

# Ejemplo de uso:
n = 3
matriz1 = np.ones((n, n))
matriz2 = np.ones((n, n)) * 2
resultado = suma_matrices_elemento_a_elemento(matriz1, matriz2)
print("Matriz resultante:")
print(resultado)
```

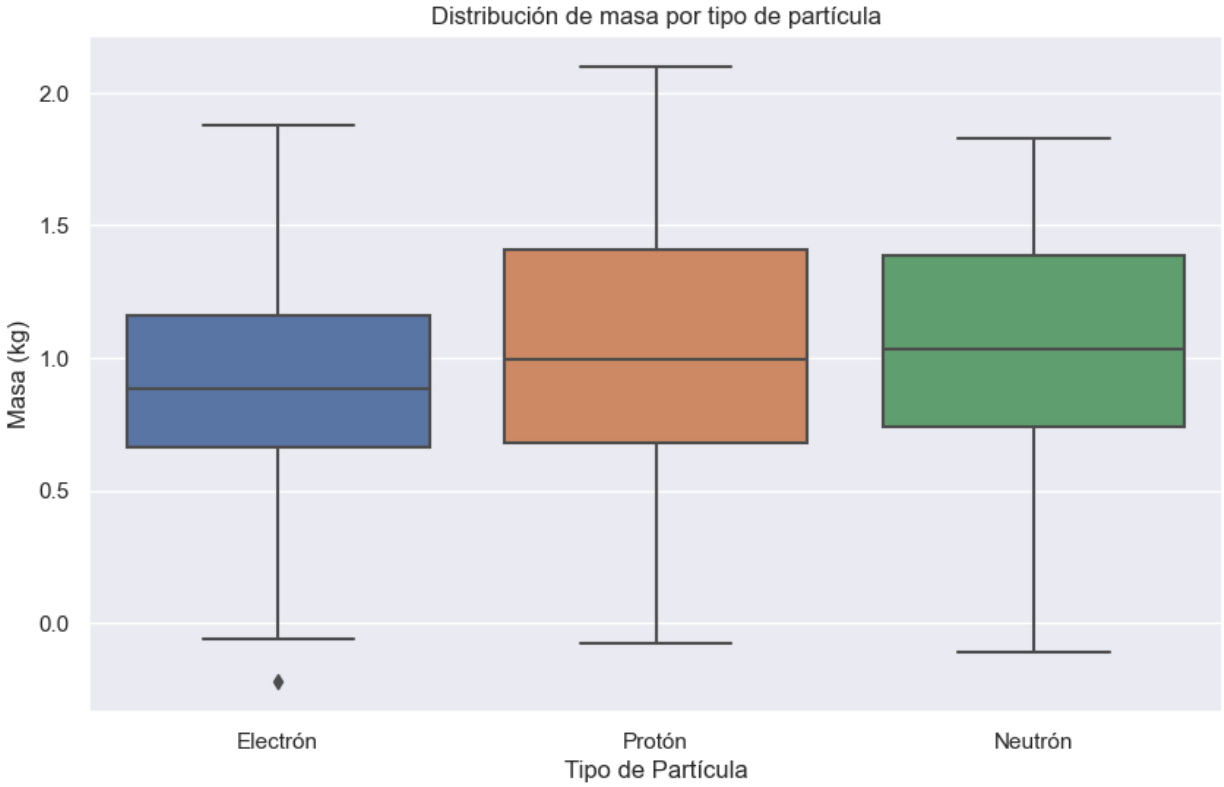
```
Matriz resultante:
[[3. 3. 3.]
 [3. 3. 3.]
 [3. 3. 3.]]
```

5.- Utiliza `Matplotlib` y `Seaborn` para trazar un gráfico de caja y bigotes mostrando la distribución de una variable numérica en función de una variable categórica en un DataFrame (usar el dataframe `datos`).

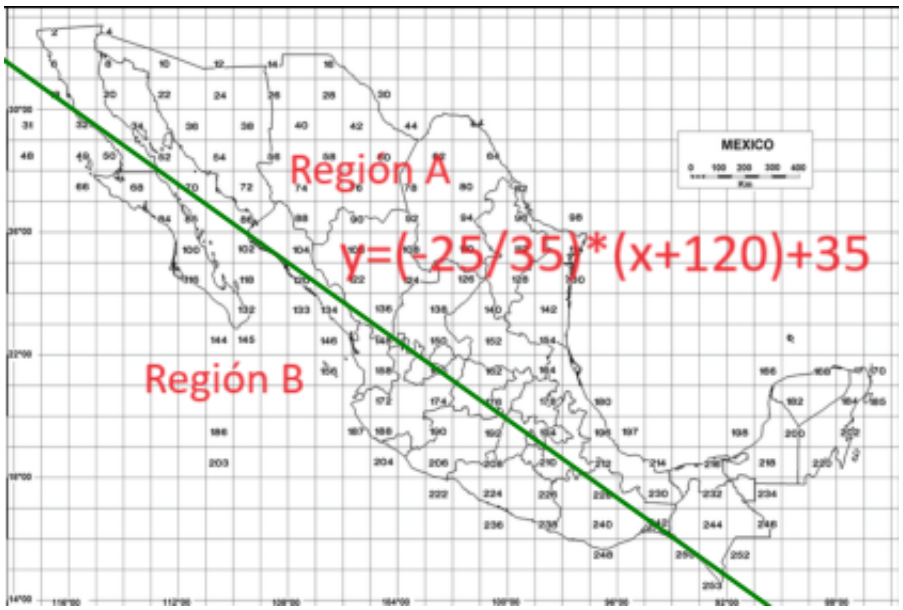
In [65]:
```
import pandas as pd
import numpy as np

# Crear un DataFrame de ejemplo
np.random.seed(0)
datos = pd.DataFrame({
    'Tipo de Partícula': np.random.choice(['Electrón', 'Protón', 'Neut
    'Masa (kg)': np.random.normal(loc=1.0, scale=0.5, size=100)  # Dis
})

# Mostrar los primeros registros del DataFrame
print(datos.head())
```

```
  Tipo de Partícula  Masa (kg)
0          Electrón   0.656705
1            Protón   1.007437
2          Electrón   0.812167
3            Protón   0.980888
4            Protón   1.183987
```

In [66]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Gráfico de caja y bigotes
plt.figure(figsize=(10, 6))
sns.boxplot(x='Tipo de Partícula', y='Masa (kg)', data=datos)
plt.xlabel('Tipo de Partícula')
plt.ylabel('Masa (kg)')
plt.title('Distribución de masa por tipo de partícula')
plt.show()
```



6.-
Descarga la base de datos de sismos en Mexico (http://www2.ssn.unam.mx:8080/catalogo/ (http://www2.ssn.unam.mx:8080/catalogo/)) desde 1985 y quita los registros con datos faltantes ("No calculable") del atributo "Magnitud". Asi tambien, realiza un diagrama de dispersion (Latitud, Longitud) para identificar las dos zonas sismicas en el mapa de Mexico (ver imagen) y realiza un filtro para ambas zonas (Region A y region B). Calcula promedio, desviacion estandar y los quartiles para ambas regiones.



In [30]:
```
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
data=pd.read_csv("SSNMX_catalogo_19850101_20240115.csv",skiprows=4,ski
data["Magnitud"]=pd.to_numeric(data["Magnitud"],errors="coerce")
data.dropna(inplace=True)
fig,ax=plt.subplots()
ax.scatter(data["Longitud"],data["Latitud"])
x=np.linspace(min(data["Longitud"]),max(data["Longitud"]),2)
ax.plot(x,(-25/35)*(x+120)+35,color="green")
regionA=data[data["Latitud"]>(-25/35)*(data["Longitud"]+120)+35]
regionB=data[data["Longitud"]<(-25/35)*(data["Longitud"]+120)+35]
ax.text(-105, 30, 'RegionA', fontsize = 18)
ax.text(-115, 20, 'RegionB', fontsize = 18)
regionA.describe()
```

```
regionB.describe()
```

Out[30]:

	Magnitud	Latitud	Longitud
count	276149.000000	276149.000000	276149.000000
mean	3.633050	17.797811	-99.033390
std	0.387014	3.999869	5.607151
min	0.300000	10.271000	-120.595000
25%	3.400000	15.930000	-101.035000
50%	3.600000	16.501300	-98.053700
75%	3.800000	18.030000	-94.936000
max	8.200000	38.092300	-85.546700