# Desafío Empresarial: Gestionando Datos de Ventas con Clases de Python

En este ejercicio, exploraremos cómo las clases de Python pueden usarse para gestionar y analizar datos de ventas. Imagina que trabajas para una pequeña empresa de venta al por menor, y tu tarea es organizar y analizar información de ventas para varios productos. Para lograr esto, crearemos una clase de Python llamada `Producto` para representar productos individuales y otra clase llamada `DatosDeVentas` para gestionar los datos de ventas de manera eficiente.

## Parte 1: Crear la Clase Producto

1. Define una clase llamada `Producto` con los siguientes atributos y métodos:
   - **Atributos**:
     - `nombre` (str): El nombre del producto.
     - `precio` (float): El precio del producto.
     - `cantidad_vendida` (int): La cantidad del producto vendido.
   - **Métodos**:
     - `__init__(self, nombre, precio)`: El método constructor que inicializa `nombre`, `precio`, y establece `cantidad_vendida` a 0.
     - `vender(self, cantidad)`: Un método que incrementa el atributo `cantidad_vendida` por la cantidad dada cuando se vende un producto.
     - `obtener_ingresos(self)`: Un método que calcula y devuelve los ingresos totales generados por la venta de este producto (precio * cantidad_vendida).

## Parte 2: Crear la Clase DatosDeVentas

1. Define una clase llamada `DatosDeVentas` con los siguientes atributos y métodos:
   - **Atributos**:
     - `productos` (lista): Una lista para almacenar instancias de la clase `Producto`.
   - **Métodos**:
     - `__init__(self)`: El método constructor que inicializa una lista vacía para almacenar productos.
     - `agregar_producto(self, producto)`: Un método que agrega un objeto `Producto` a la lista de productos.
     - `obtener_ingresos_totales(self)`: Un método que calcula y devuelve los ingresos totales generados por la venta de todos los productos.
     - `obtener_producto_mas_vendido(self)`: Un método que identifica y devuelve el producto con la mayor cantidad vendida.

## Parte 3: Usando las Clases

1. Crea instancias de la clase `Producto` para representar diferentes productos en tu inventario.

2. Crea una instancia de la clase `DatosDeVentas` para gestionar tus datos de ventas.
3. Añade las instancias de producto creadas a la instancia de `DatosDeVentas`.
4. Simula ventas para cada producto usando el método `vender` de la clase `Producto`.
5. Calcula los ingresos totales usando el método `obtener_ingresos_totales` de la clase `DatosDeVentas`.
6. Identifica el producto más vendido usando el método `obtener_producto_mas_vendido` de la clase `DatosDeVentas`.

## Desafíos Adicionales (Opcional, solo si tienes tiempo):

1. Implementa manejo de errores en tus clases para asegurar que cantidades negativas o entradas incorrectas sean manejadas de manera adecuada.
2. Crea un método para mostrar los detalles del producto, incluyendo nombre, precio, cantidad vendida e ingresos, para todos los productos en la clase `DatosDeVentas`.

Este ejercicio te ayudará a entender cómo las clases pueden usarse para organizar y analizar datos de ventas de manera efectiva, facilitando la gestión y extracción de información valiosa de tus datos. Demuestra la aplicación práctica de las clases en un contexto de análisis de datos.

```
# Parte 1: Crear la clase Producto
class Producto:
    def __init__(self, nombre, precio):
        # Validación de nombre
        if not isinstance(nombre, str) or not nombre.strip():
            raise ValueError("El nombre del producto debe ser una
cadena de texto no vacía.")

        # Validación de precio
        if not isinstance(precio, (int, float)) or precio <= 0:
            raise ValueError("El precio debe ser un número positivo.")

        self.nombre = nombre
        self.precio = precio
        self.cantidad_vendida = 0  # Inicializa la cantidad vendida en
0

    def vender(self, cantidad):
        # Validación de cantidad vendida
        if not isinstance(cantidad, int) or cantidad <= 0:
            raise ValueError("La cantidad vendida debe ser un entero
positivo.")
        self.cantidad_vendida += cantidad

    def obtener_ingresos(self):
        # Calcula los ingresos totales
        return self.precio * self.cantidad_vendida
```

```
    def mostrar_detalles(self):
        # Devuelve los detalles del producto
        return f"Producto: {self.nombre}, Precio: {self.precio},
Cantidad Vendida: {self.cantidad_vendida}, Ingresos:
{self.obtener_ingresos()}"

# Parte 2: Crear la clase DatosDeVentas
from functools import reduce

class DatosDeVentas:
    def __init__(self):
        self.productos = []  # Inicializa la lista de productos

    def agregar_producto(self, producto):
        # Validación de que se está agregando un objeto Producto
        if not isinstance(producto, Producto):
            raise ValueError("Solo se pueden agregar instancias de la
clase Producto.")
        self.productos.append(producto)

    def obtener_ingresos_totales(self):
        # Usa map y reduce para calcular los ingresos totales
        ingresos = map(lambda producto: producto.obtener_ingresos(),
self.productos)
        return reduce(lambda total, ingreso: total + ingreso,
ingresos, 0)

    def obtener_producto_mas_vendido(self):
        # Verifica si hay productos en la lista
        if not self.productos:
            return None
        # Encuentra el producto con la mayor cantidad vendida
        return max(self.productos, key=lambda producto:
producto.cantidad_vendida)

    def mostrar_todos_los_productos(self):
        # Usa map para mostrar los detalles de cada producto
        detalles = map(lambda producto: producto.mostrar_detalles(),
self.productos)
        return "\n".join(detalles)

# 3. Crear instancias de la clase Producto
producto1 = Producto("Laptop", 800.0)
producto2 = Producto("Teléfono", 500.0)
producto3 = Producto("Tablet", 300.0)

# 4. Crear instancia de DatosDeVentas
datos_de_ventas = DatosDeVentas()
```

```python
# 5. Añadir productos a la instancia de DatosDeVentas
datos_de_ventas.agregar_producto(producto1)
datos_de_ventas.agregar_producto(producto2)
datos_de_ventas.agregar_producto(producto3)

# 6. Simular ventas
producto1.vender(10)  # Laptop: 10 unidades
producto2.vender(15)  # Teléfono: 15 unidades
producto3.vender(5)   # Tablet: 5 unidades

# 7. Calcular ingresos totales
ingresos_totales = datos_de_ventas.obtener_ingresos_totales()
print(f"Ingresos Totales: ${ingresos_totales}")
```

```
Ingresos Totales: $17000.0
```

```python
# 8. Identificar el producto más vendido
producto_mas_vendido = datos_de_ventas.obtener_producto_mas_vendido()
if producto_mas_vendido:
    print(f"Producto más vendido: {producto_mas_vendido.nombre} con
{producto_mas_vendido.cantidad_vendida} unidades vendidas")
```

```
Producto más vendido: Teléfono con 15 unidades vendidas
```

```python
# 9. Mostrar detalles de todos los productos
print("\nDetalles de los productos vendidos:")
print(datos_de_ventas.mostrar_todos_los_productos())
```

```

Detalles de los productos vendidos:
Producto: Laptop, Precio: 800.0, Cantidad Vendida: 10, Ingresos:
8000.0
Producto: Teléfono, Precio: 500.0, Cantidad Vendida: 15, Ingresos:
7500.0
Producto: Tablet, Precio: 300.0, Cantidad Vendida: 5, Ingresos: 1500.0
```

# Desafío: Creando el Arca de Noé con Clases de Python

En este ejercicio, desarrollaremos un modelo del famoso Arca de Noé utilizando programación orientada a objetos en Python. Crearemos una clase estática `Arca` que tendrá la capacidad de almacenar tanto animales como alimentos y agua. La arca tendrá contenedores específicos y finitos, y proporcionará métodos para gestionar los alimentos y cuidar de los animales, incluyendo alimentarlos y darles agua.

## Parte 1: Crear la Clase Arca

1. Define una clase estática llamada `Arca` con los siguientes atributos y métodos:
   - **Atributos**:
     - `animales` (lista): Una lista para almacenar instancias de la clase `Animal`.
     - `alimentos` (lista): Una lista para almacenar instancias de la clase `Alimento`.

- `agua` (int): Un contenedor que almacena la cantidad total de agua disponible.
- `capacidad_maxima` (int): Un límite para la cantidad total de animales y alimentos que puede contener el arca.

– **Métodos**:
- `__init__(cls, capacidad_maxima)`: El método constructor que inicializa la capacidad máxima del arca y crea listas vacías para animales y alimentos, además de establecer el agua a un valor inicial.
- `agregar_animal(cls, animal)`: Un método que agrega un objeto `Animal` a la lista de animales si no se supera la capacidad máxima.
- `agregar_alimento(cls, alimento)`: Un método que agrega un objeto `Alimento` a la lista de alimentos.
- `agregar_agua(cls, cantidad)`: Un método que agrega agua al contenedor de agua.
- `alimentar_animal(cls, animal)`: Un método que proporciona alimento a un animal específico.
- `dar_agua(cls, animal)`: Un método que proporciona agua a un animal específico.
- `estado_arca(cls)`: Un método estático que devuelve el estado actual de la arca, como el número de animales, alimentos y la cantidad de agua almacenados.

## Parte 2: Crear las Clases Padre Animal y Alimento

1. Define una clase llamada `Animal` con los siguientes atributos y métodos:
   – **Atributos**:
   - `nombre` (str): El nombre del animal.
   - `tipo` (str): El tipo de animal (por ejemplo, "perro", "gato").
   - `hambre` (int): Un nivel que indica cuánta hambre tiene el animal.
   - `sed` (int): Un nivel que indica cuánta sed tiene el animal.

   – **Métodos**:
   - `__init__(self, nombre, tipo)`: El método constructor que inicializa el nombre y tipo del animal, y establece hambre y sed a un valor inicial.
   - `alimentar(self)`: Un método que reduce el nivel de hambre del animal.
   - `dar_agua(self)`: Un método que reduce el nivel de sed del animal.
   - `estado(self)`: Un método que devuelve el estado actual de hambre y sed del animal.
2. Define una clase llamada `Alimento` con los siguientes atributos y métodos:
   – **Atributos**:
   - `tipo` (str): El tipo de alimento (por ejemplo, "heno", "croquetas").
   - `cantidad` (int): La cantidad de alimento disponible.

   – **Métodos**:
   - `__init__(self, tipo, cantidad)`: El método constructor que inicializa el tipo de alimento y la cantidad.

- `usar(self, cantidad)`: Un método que reduce la cantidad de alimento disponible en la cantidad especificada.
- `es_alimento_adecuado(cls, tipo_animal)`: Un método estático que verifica si un tipo de alimento es adecuado para un tipo de animal dado.

## Parte 3: Crear Clases Derivadas

1. Crea clases derivadas de `Animal` para diferentes tipos de animales (por ejemplo, `Perro`, `Gato`) que pueden tener métodos específicos o atributos adicionales.

2. Crea clases derivadas de `Alimento` para diferentes tipos de alimentos (por ejemplo, `Heno`, `Croquetas`) que pueden tener métodos específicos o atributos adicionales.

## Parte 4: Usando las Clases

1. Crea una instancia de la clase `Arca` con una capacidad máxima definida.

2. Crea instancias de los animales y alimentos derivados de sus respectivas clases padre.

3. Añade los animales y alimentos creados a la instancia de `Arca`.

4. Añade agua al contenedor de agua usando el método `agregar_agua`.

5. Simula el proceso de alimentar a los animales y darles agua utilizando los métodos correspondientes de la clase `Arca`.

6. Utiliza el método estático `estado_arca` para verificar el estado actual del arca.

## Desafíos Adicionales (Opcional, solo si tienes tiempo):

1. Implementa manejo de errores en tus clases para asegurar que no se puedan agregar más animales o alimentos que la capacidad máxima de la arca.

2. Crea un método para mostrar el estado de todos los animales en el arca, incluyendo su nombre, tipo, hambre y sed.

Este ejercicio te ayudará a comprender cómo usar la programación orientada a objetos para modelar un sistema más complejo, además de permitirte practicar la creación de jerarquías de clases y la interacción entre ellas. La inclusión de métodos estáticos también te dará experiencia en la implementación de funcionalidad que no depende del estado específico de una instancia.

```
from functools import reduce

# Parte 1: Definir la Clase Arca
class Arca:
    def __init__(self, capacidad_maxima, cantidad_agua=0):
        if not isinstance(capacidad_maxima, int):

            raise TypeError("La capacidad máxima debe ser un número
entero.")
```

```
        if capacidad_maxima <= 0:
            raise ValueError("La capacidad máxima debe ser un valor
positivo.")
        if not isinstance(cantidad_agua, int):
            raise TypeError("La cantidad de agua debe ser un número
entero.")
        if cantidad_agua < 0:
            raise ValueError("La cantidad de agua no puede ser
negativa.")
        if cantidad_agua > capacidad_maxima:
            raise ValueError("La cantidad de agua no puede superar la
capacidad máxima del arca.")

        self.animales = []
        self.alimentos = []
        self.agua = cantidad_agua  # Inicializar con la cantidad de
agua proporcionada
        self.capacidad_maxima = capacidad_maxima

    def capacidad_total(self):
        # Sumar la cantidad total ocupada por animales, alimentos y
agua
        return len(self.animales) + len(self.alimentos) + self.agua

    def agregar_animal(self, animal):
        if not isinstance(animal, Animal):
            raise TypeError("El objeto debe ser una instancia de la
clase Animal.")
        if self.capacidad_total() < self.capacidad_maxima:
            self.animales.append(animal)
        else:
            raise ValueError("No se puede agregar más animales,
capacidad máxima alcanzada.")

    def agregar_alimento(self, alimento):
        if not isinstance(alimento, Alimento):
            raise TypeError("El objeto debe ser una instancia de la
clase Alimento.")

        # Asegúrate de que hay capacidad suficiente
        if self.capacidad_total() + alimento.contenedores >
self.capacidad_maxima:
            raise ValueError("No se puede agregar más alimentos,
capacidad máxima alcanzada.")

        self.alimentos.append(alimento)

    def agregar_agua(self, cantidad):
        if not isinstance(cantidad, int):
            raise TypeError("La cantidad de agua debe ser un número
```

```
entero.")
        if cantidad <= 0:
            raise ValueError("La cantidad de agua debe ser un valor
positivo.")

        # Verifica la capacidad total incluyendo el agua
        if self.capacidad_total() + cantidad > self.capacidad_maxima:
            raise ValueError("No se puede agregar más agua, capacidad
máxima alcanzada.")

        self.agua += cantidad  # Agregar el número de contenedores de
agua

    def alimentar_animal(self, animal):
        if not isinstance(animal, Animal):
            raise TypeError("El objeto debe ser una instancia de la
clase Animal.")
        if animal in self.animales:
            # Busca alimento que coincida con el tipo del animal
            alimento = next((a for a in self.alimentos if a.tipo in
animal.tipo_alimento and a.cantidad >= 10), None)
            if alimento:
                animal.alimentar()
                alimento.usar(1)  # Usar 1 contenedor de alimento (10
porciones)
                print(f"{animal.nombre} ha sido alimentado. Estado -
Hambre: {animal.hambre}, Sed: {animal.sed}")
            else:
                tipos_faltantes = [at for at in animal.tipo_alimento
if not any(a.tipo == at and a.cantidad >= 10 for a in self.alimentos)]
                print(f"Error alimentando a {animal.nombre}: No hay
alimento adecuado disponible. Faltan: {', '.join(tipos_faltantes)}")
        else:
            print(f"Error alimentando a {animal.nombre}: El animal no
está en el arca.")

    def dar_agua(self, animal):
        if not isinstance(animal, Animal):
            raise TypeError("El objeto debe ser una instancia de la
clase Animal.")
        if animal in self.animales:
            # Verificar si hay suficientes porciones de agua
            if self.agua > 0:
                animal.dar_agua()
                # Usar un contenedor de agua (10 porciones)
                self.agua -= 1
                print(f"{animal.nombre} ha recibido agua. Estado -
Hambre: {animal.hambre}, Sed: {animal.sed}")
            else:
```

```
                print(f"Error dando agua a {animal.nombre}: No hay
agua disponible.")
        else:
            print(f"Error dando agua a {animal.nombre}: El animal no
está en el arca.")

    @staticmethod
    def estado_arca(arca):
        if not isinstance(arca, Arca):
            raise TypeError("El argumento debe ser una instancia de la
clase Arca.")

        # Contar los contenedores y las porciones de alimento
        contenedores_alimentos = reduce(lambda x, y: x + y, map(lambda
a: a.contenedores, arca.alimentos), 0)
        porciones_alimentos = reduce(lambda x, y: x + y, map(lambda a:
a.cantidad, arca.alimentos), 0)  # Total de porciones

        # Calcular el número de contenedores de agua y porciones
        contenedores_agua = arca.agua
        porciones_agua = contenedores_agua * 10  # Cada contenedor
tiene 10 porciones

        return {
            "Número de animales": len(arca.animales),
            "Número de contenedores de alimentos":
contenedores_alimentos,
            "Porciones de alimentos": porciones_alimentos,
            "Número de contenedores de agua": contenedores_agua,
            "Porciones de agua": porciones_agua,
            "Capacidad máxima de contenedores": arca.capacidad_maxima
# Cambio aquí
        }

# Parte 2: Definir la Clase Padre Animal
class Animal:
    def __init__(self, nombre, tipo, tipo_alimento):
        if not isinstance(nombre, str):
            raise TypeError("El nombre debe ser una cadena de texto.")
        if not isinstance(tipo, str):
            raise TypeError("El tipo debe ser una cadena de texto.")
        if not isinstance(tipo_alimento, list):
            raise TypeError("El tipo de alimento debe ser una lista.")
        if not all(isinstance(item, str) for item in tipo_alimento):
            raise TypeError("Todos los elementos en la lista de tipo
de alimento deben ser cadenas de texto.")

        self.nombre = nombre
        self.tipo = tipo
        self.tipo_alimento = tipo_alimento  # Lista de tipos de
```

```
alimentos que consume
        self.hambre = 5  # Inicialmente, nivel de hambre
        self.sed = 5     # Inicialmente, nivel de sed

    def alimentar(self):
        if self.hambre > 0:
            self.hambre -= 1  # Reduce el hambre en 1

    def dar_agua(self):
        if self.sed > 0:
            self.sed -= 1  # Reduce la sed en 1

    def estado(self):
        return {
            "Nombre": self.nombre,
            "Tipo": self.tipo,
            "Tipo de Alimento": self.tipo_alimento,
            "Hambre": self.hambre,
            "Sed": self.sed
        }

# Parte 3: Definir la Clase Alimento
class Alimento:
    def __init__(self, tipo, contenedores):
        if not isinstance(tipo, str):
            raise TypeError("El tipo debe ser una cadena de texto.")
        if not isinstance(contenedores, int) or contenedores <= 0:
            raise ValueError("La cantidad de contenedores debe ser un
número entero positivo.")

        self.tipo = tipo
        self.cantidad = contenedores * 10  # Cada contenedor tiene 10
porciones
        self.contenedores = contenedores

    def usar(self, cantidad):
        if not isinstance(cantidad, int):
            raise TypeError("La cantidad a usar debe ser un número
entero.")
        if cantidad <= 0 or cantidad > self.contenedores:
            raise ValueError("La cantidad a usar debe ser un valor
positivo y no puede superar el número de contenedores.")
        self.cantidad -= cantidad * 10  # Reducir porciones
        self.contenedores -= cantidad  # Reducir contenedores

# Parte 4: Definir Clases Derivadas para diferentes tipos de Alimento
class Carne(Alimento):
    def __init__(self, contenedores):
        if not isinstance(contenedores, int) or contenedores <= 0:
            raise ValueError("La cantidad de contenedores debe ser un
```

```
número entero positivo.")
        super().__init__("carne", contenedores)

class Pescado(Alimento):
    def __init__(self, contenedores):
        if not isinstance(contenedores, int) or contenedores <= 0:
            raise ValueError("La cantidad de contenedores debe ser un
número entero positivo.")
        super().__init__("pescado", contenedores)

class Croquetas(Alimento):
    def __init__(self, contenedores):
        if not isinstance(contenedores, int) or contenedores <= 0:
            raise ValueError("La cantidad de contenedores debe ser un
número entero positivo.")
        super().__init__("croquetas", contenedores)

# Parte 5: Definir Clases Derivadas para diferentes tipos de Animales
class Felino(Animal):
    def __init__(self, nombre):
        if not isinstance(nombre, str):
            raise TypeError("El nombre debe ser una cadena de texto.")
        super().__init__(nombre, "Felino", ["carne", "pescado"])

class Canino(Animal):
    def __init__(self, nombre):
        if not isinstance(nombre, str):
            raise TypeError("El nombre debe ser una cadena de texto.")
        super().__init__(nombre, "Canino", ["carne", "croquetas"])

class Tigre(Felino):
    def __init__(self, nombre):
        if not isinstance(nombre, str):
            raise TypeError("El nombre debe ser una cadena de texto.")
        super().__init__(nombre)

class Leon(Felino):
    def __init__(self, nombre):
        if not isinstance(nombre, str):
            raise TypeError("El nombre debe ser una cadena de texto.")
        super().__init__(nombre)

class Pantera(Felino):
    def __init__(self, nombre):
        if not isinstance(nombre, str):
            raise TypeError("El nombre debe ser una cadena de texto.")
        super().__init__(nombre)

class Lobo(Canino):
    def __init__(self, nombre):
```

```
        if not isinstance(nombre, str):
            raise TypeError("El nombre debe ser una cadena de texto.")
        super().__init__(nombre)

class Zorro(Canino):
    def __init__(self, nombre):
        if not isinstance(nombre, str):
            raise TypeError("El nombre debe ser una cadena de texto.")
        super().__init__(nombre)

class Perro(Canino):
    def __init__(self, nombre):
        if not isinstance(nombre, str):
            raise TypeError("El nombre debe ser una cadena de texto.")
        super().__init__(nombre)

# Parte 6: Crear instancia de Arca
arca = Arca(capacidad_maxima=100)

# Parte 7: Crear instancias de los animales y alimentos derivados de
sus respectivas clases padre usando comprensión de listas
nombres_felinos = [f"Tigre{i+1}" for i in range(3)] + [f"León{i+1}"
for i in range(3)] + [f"Pantera{i+1}" for i in range(3)]
nombres_caninos = [f"Lobo{i+1}" for i in range(3)] + [f"Zorro{i+1}"
for i in range(3)] + [f"Perro{i+1}" for i in range(3)]

# Crear lista de animales usando map
animales_felinos = [Tigre(nombre) for nombre in nombres_felinos[:3]] +
[Leon(nombre) for nombre in nombres_felinos[3:6]] + [Pantera(nombre)
for nombre in nombres_felinos[6:]]
animales_caninos = [Lobo(nombre) for nombre in nombres_caninos[:3]] +
[Zorro(nombre) for nombre in nombres_caninos[3:6]] + [Perro(nombre)
for nombre in nombres_caninos[6:]]

# Unir listas de animales
animales = animales_felinos + animales_caninos

# Usando filter para crear una lista de alimentos válidos
alimentos = [
    Carne(13),   # 13 contenedores de carne (130 porciones)
    Pescado(10), # 10 contenedores de pescado (100 porciones)
    Croquetas(10) # 10 contenedores de croquetas (100 porciones)
]

# Parte 8: Añadir animales y alimentos creados a la instancia de Arca
for animal in animales:
    try:
        arca.agregar_animal(animal)
    except ValueError as e:
        print(f"Error al agregar a {animal.nombre}: {e}")
```

```python
# Agregar agua
try:
    arca.agregar_agua(10)
except ValueError as e:
    print(f"Error al agregar agua: {e}")

# Agregar alimentos
for alimento in alimentos:
    try:
        arca.agregar_alimento(alimento)
    except ValueError as e:
        print(f"Error al agregar {alimento.tipo}: {e}")

# Parte 9: Mostrar el estado del arca
estado = Arca.estado_arca(arca)
print("Estado del arca:", estado)
```

```
Estado del arca: {'Número de animales': 18, 'Número de contenedores de alimentos': 33, 'Porciones de alimentos': 330, 'Número de contenedores de agua': 10, 'Porciones de agua': 100, 'Capacidad máxima de contenedores': 100}
```

```python
# Parte 10: Simular el proceso de alimentar a los animales y darles agua
for animal in animales:
    try:
        arca.alimentar_animal(animal)
        arca.dar_agua(animal)
    except ValueError as e:
        print(f"Error alimentando a {animal.nombre}: {e}")
```

```
Tigre1 ha sido alimentado. Estado - Hambre: 4, Sed: 5
Tigre1 ha recibido agua. Estado - Hambre: 4, Sed: 4
Tigre2 ha sido alimentado. Estado - Hambre: 4, Sed: 5
Tigre2 ha recibido agua. Estado - Hambre: 4, Sed: 4
Tigre3 ha sido alimentado. Estado - Hambre: 4, Sed: 5
Tigre3 ha recibido agua. Estado - Hambre: 4, Sed: 4
León1 ha sido alimentado. Estado - Hambre: 4, Sed: 5
León1 ha recibido agua. Estado - Hambre: 4, Sed: 4
León2 ha sido alimentado. Estado - Hambre: 4, Sed: 5
León2 ha recibido agua. Estado - Hambre: 4, Sed: 4
León3 ha sido alimentado. Estado - Hambre: 4, Sed: 5
León3 ha recibido agua. Estado - Hambre: 4, Sed: 4
Pantera1 ha sido alimentado. Estado - Hambre: 4, Sed: 5
Pantera1 ha recibido agua. Estado - Hambre: 4, Sed: 4
Pantera2 ha sido alimentado. Estado - Hambre: 4, Sed: 5
Pantera2 ha recibido agua. Estado - Hambre: 4, Sed: 4
Pantera3 ha sido alimentado. Estado - Hambre: 4, Sed: 5
Pantera3 ha recibido agua. Estado - Hambre: 4, Sed: 4
Lobo1 ha sido alimentado. Estado - Hambre: 4, Sed: 5
```

```
Lobo1 ha recibido agua. Estado - Hambre: 4, Sed: 4
Lobo2 ha sido alimentado. Estado - Hambre: 4, Sed: 5
Error dando agua a Lobo2: No hay agua disponible.
Lobo3 ha sido alimentado. Estado - Hambre: 4, Sed: 5
Error dando agua a Lobo3: No hay agua disponible.
Zorro1 ha sido alimentado. Estado - Hambre: 4, Sed: 5
Error dando agua a Zorro1: No hay agua disponible.
Zorro2 ha sido alimentado. Estado - Hambre: 4, Sed: 5
Error dando agua a Zorro2: No hay agua disponible.
Zorro3 ha sido alimentado. Estado - Hambre: 4, Sed: 5
Error dando agua a Zorro3: No hay agua disponible.
Perro1 ha sido alimentado. Estado - Hambre: 4, Sed: 5
Error dando agua a Perro1: No hay agua disponible.
Perro2 ha sido alimentado. Estado - Hambre: 4, Sed: 5
Error dando agua a Perro2: No hay agua disponible.
Perro3 ha sido alimentado. Estado - Hambre: 4, Sed: 5
Error dando agua a Perro3: No hay agua disponible.
```

```
# Parte 11: Mostrar el estado del arca después de las simulaciones
estado_final = Arca.estado_arca(arca)
print("Estado final del arca:", estado_final)
```

```
Estado final del arca: {'Número de animales': 18, 'Número de
contenedores de alimentos': 15, 'Porciones de alimentos': 150, 'Número
de contenedores de agua': 0, 'Porciones de agua': 0, 'Capacidad máxima
de contenedores': 100}
```

```
# Parte 12: Mostrar el estado de todos los animales en el arca
for animal in arca.animales:
    print(animal.estado())
```

```
{'Nombre': 'Tigre1', 'Tipo': 'Felino', 'Tipo de Alimento': ['carne',
'pescado'], 'Hambre': 4, 'Sed': 4}
{'Nombre': 'Tigre2', 'Tipo': 'Felino', 'Tipo de Alimento': ['carne',
'pescado'], 'Hambre': 4, 'Sed': 4}
{'Nombre': 'Tigre3', 'Tipo': 'Felino', 'Tipo de Alimento': ['carne',
'pescado'], 'Hambre': 4, 'Sed': 4}
{'Nombre': 'León1', 'Tipo': 'Felino', 'Tipo de Alimento': ['carne',
'pescado'], 'Hambre': 4, 'Sed': 4}
{'Nombre': 'León2', 'Tipo': 'Felino', 'Tipo de Alimento': ['carne',
'pescado'], 'Hambre': 4, 'Sed': 4}
{'Nombre': 'León3', 'Tipo': 'Felino', 'Tipo de Alimento': ['carne',
'pescado'], 'Hambre': 4, 'Sed': 4}
{'Nombre': 'Pantera1', 'Tipo': 'Felino', 'Tipo de Alimento': ['carne',
'pescado'], 'Hambre': 4, 'Sed': 4}
{'Nombre': 'Pantera2', 'Tipo': 'Felino', 'Tipo de Alimento': ['carne',
'pescado'], 'Hambre': 4, 'Sed': 4}
{'Nombre': 'Pantera3', 'Tipo': 'Felino', 'Tipo de Alimento': ['carne',
'pescado'], 'Hambre': 4, 'Sed': 4}
{'Nombre': 'Lobo1', 'Tipo': 'Canino', 'Tipo de Alimento': ['carne',
```

```
'croquetas'], 'Hambre': 4, 'Sed': 4}
{'Nombre': 'Lobo2', 'Tipo': 'Canino', 'Tipo de Alimento': ['carne',
'croquetas'], 'Hambre': 4, 'Sed': 5}
{'Nombre': 'Lobo3', 'Tipo': 'Canino', 'Tipo de Alimento': ['carne',
'croquetas'], 'Hambre': 4, 'Sed': 5}
{'Nombre': 'Zorro1', 'Tipo': 'Canino', 'Tipo de Alimento': ['carne',
'croquetas'], 'Hambre': 4, 'Sed': 5}
{'Nombre': 'Zorro2', 'Tipo': 'Canino', 'Tipo de Alimento': ['carne',
'croquetas'], 'Hambre': 4, 'Sed': 5}
{'Nombre': 'Zorro3', 'Tipo': 'Canino', 'Tipo de Alimento': ['carne',
'croquetas'], 'Hambre': 4, 'Sed': 5}
{'Nombre': 'Perro1', 'Tipo': 'Canino', 'Tipo de Alimento': ['carne',
'croquetas'], 'Hambre': 4, 'Sed': 5}
{'Nombre': 'Perro2', 'Tipo': 'Canino', 'Tipo de Alimento': ['carne',
'croquetas'], 'Hambre': 4, 'Sed': 5}
{'Nombre': 'Perro3', 'Tipo': 'Canino', 'Tipo de Alimento': ['carne',
'croquetas'], 'Hambre': 4, 'Sed': 5}
```