Coordinación de
Educación Abierta y a Distancia
VICERRECTORADO ACADÉMICO

Unach
UNIVERSIDAD NACIONAL DE CHIMBORAZO Online

PROGRAMACIÓN II

Actividad Autónoma 2: Clasificación del Data set Iris utilizando

Programación Orientada a Objetos (POO) y Python

Unidad 1: Fundamentos de la Programación Orientada a Objetos (POO)

Tema 2: Herencia y Abstracción

FACULTAD DE
Ingeniería

Nombres: Lyndon Andrés Castro Vaca

Fecha: 11-05-2025

Carrera: Ciencia de Datos e Inteligencia Artificial

Periodo académico: 2025

Semestre: 2do “B”

Objetivo de la actividad: Aplicar los conceptos de herencia, polimorfismo, métodos mágicos y clases abstractas aprendidos en las semanas 3 y 4, mediante la implementación de un proyecto que utilice los principios de la Programación Orientada a Objetos para resolver un problema de clasificación con el data set Iris, enfocándose en la modularidad, extensibilidad y reutilización de código.

Recursos o temas que debe haber estudiado antes de hacer la actividad:

• Conceptos fundamentales de POO: clases, objetos, herencia.

• Definición de atributos y métodos en Python.

• Métodos mágicos (__str__, __add__) y sobrecarga de operadores.

• Clases abstractas y polimorfismo.

Formato de entrega: Formato PDF, generado en Jupyter Notebook (máximo 5MB).
Instrucciones:

1. **Definir la clase Flor:**
   - Atributos privados: longitud_sepalo, ancho_sepalo, longitud_petalo, ancho_petalo, especie.
   - Método __init__ para inicializar los atributos.
   - Métodos getters y setters para controlar el acceso y la modificación de atributos privados.
   - Método __str__ para mostrar la información de la flor.
   - Añadir un método calcular_volumen_flor que calcule un volumen estimado utilizando los atributos.
2. **Crear objetos Flor:**
   - Crear al menos cinco instancias de Flor con datos del dataset Iris.
   - Utilizar los métodos setters para modificar algunos atributos y mostrar los cambios utilizando __str__.
3. **Implementar la clase abstracta Clasificador:**
   - Contener un método abstracto clasificar_por_ancho().
4. **Crear la subclase ClasificadorIris:**

   Heredar de Clasificador e implementar clasificar_por_ancho para clasificar las flores según el ancho del sépalo.

   - Mostrar los resultados en un diccionario con anchos agrupados en rangos.
5. **Sobrecarga de operadores:**
   • Sobrecargar el operador + para crear una flor promedio combinando dos instancias de Flor.
   • Sobrecargar el operador * para escalar todos los atributos numéricos de una flor.
6. **Comparación de distancias:**
   • En la clase Flor, implementar los métodos:
      o Distancia de Manhattan.
      o Distancia de Chebyshev.
   • Comparar distancias entre diferentes instancias y determinar cuál está más cerca de una flor de referencia.

7. **Visualización de resultados:**

• Utilizar matplotlib para graficar la distribución de las flores según el ancho del sépalo.
• Mostrar un gráfico de barras comparando los volúmenes calculados de las flores.

8. **Documentación:**
   • Incluir comentarios explicativos en cada sección del código

```python
# Para Importar las librerías necesarias
import matplotlib.pyplot as plt
from abc import ABC, abstractmethod
from math import sqrt

# 1. Defino la clase Flor con sus atributos y métodos
class Flor:
    def __init__(self, longitud_sepalo, ancho_sepalo, longitud_petalo, ancho_petalo, especie):
        """Inicializo los atributos privados de la flor"""
        self.__longitud_sepalo = longitud_sepalo
        self.__ancho_sepalo = ancho_sepalo
        self.__longitud_petalo = longitud_petalo
        self.__ancho_petalo = ancho_petalo
        self.__especie = especie

    # Métodos getter para acceder a los atributos
    @property
    def longitud_sepalo(self):
        return self.__longitud_sepalo

    @property
    def ancho_sepalo(self):
        return self.__ancho_sepalo

    @property
    def longitud_petalo(self):
        return self.__longitud_petalo

    @property
    def ancho_petalo(self):
        return self.__ancho_petalo

    @property
    def especie(self):
        return self.__especie

    # Métodos setter para modificar los atributos
    @longitud_sepalo.setter
    def longitud_sepalo(self, valor):
        self.__longitud_sepalo = valor
```

```
41.
42.     @ancho_sepalo.setter
43.     def ancho_sepalo(self, valor):
44.         self.__ancho_sepalo = valor
45.
46.     @longitud_petalo.setter
47.     def longitud_petalo(self, valor):
48.         self.__longitud_petalo = valor
49.
50.     @ancho_petalo.setter
51.     def ancho_petalo(self, valor):
52.         self.__ancho_petalo = valor
53.
54.     @especie.setter
55.     def especie(self, valor):
56.         self.__especie = valor
57.
58.     def __str__(self):
59.         """Creo una cadena legible que describa la flor"""
60.         return (f"Flor: {self.__especie}\n"
61.                 f"Longitud sépalo: {self.__longitud_sepalo} cm\n"
62.                 f"Ancho sépalo: {self.__ancho_sepalo} cm\n"
63.                 f"Longitud pétalo: {self.__longitud_petalo} cm\n"
64.                 f"Ancho pétalo: {self.__ancho_petalo} cm")
65.
66.     def calcular_volumen_flor(self):
67.         """Estimo el volumen multiplicando todas las dimensiones"""
68.         return self.__longitud_sepalo * self.__ancho_sepalo *
    self.__longitud_petalo * self.__ancho_petalo
69.
70.     def distancia_manhattan(self, otra_flor):
71.         """Calculo la distancia Manhattan entre esta flor y otra"""
72.         return (abs(self.__longitud_sepalo - otra_flor.longitud_sepalo) +
73.                 abs(self.__ancho_sepalo - otra_flor.ancho_sepalo) +
74.                 abs(self.__longitud_petalo - otra_flor.longitud_petalo) +
75.                 abs(self.__ancho_petalo - otra_flor.ancho_petalo))
76.
77.     def distancia_chebyshev(self, otra_flor):
78.         """Calculo la distancia Chebyshev (la mayor diferencia entre
    atributos)"""
79.         dif_long_sep = abs(self.__longitud_sepalo - otra_flor.longitud_sepalo)
80.         dif_anch_sep = abs(self.__ancho_sepalo - otra_flor.ancho_sepalo)
81.         dif_long_pet = abs(self.__longitud_petalo - otra_flor.longitud_petalo)
82.         dif_anch_pet = abs(self.__ancho_petalo - otra_flor.ancho_petalo)
83.
84.         return max(dif_long_sep, dif_anch_sep, dif_long_pet, dif_anch_pet)
85.
86.     # Sobrecargo el operador + para crear una flor "promedio"
87.     def __add__(self, otra_flor):
```

```
        """Promedio los atributos de dos flores y creo una nueva"""
        long_sep = (self.__longitud_sepalo + otra_flor.longitud_sepalo) / 2
        anch_sep = (self.__ancho_sepalo + otra_flor.ancho_sepalo) / 2
        long_pet = (self.__longitud_petalo + otra_flor.longitud_petalo) / 2
        anch_pet = (self.__ancho_petalo + otra_flor.ancho_petalo) / 2
        especie = f"Híbrido {self.__especie}-{otra_flor.especie}"

        return Flor(long_sep, anch_sep, long_pet, anch_pet, especie)

    # Sobrecargo el operador * para escalar los atributos numéricos
    def __mul__(self, escalar):
        """Escalo la flor multiplicando sus dimensiones por un escalar"""
        long_sep = self.__longitud_sepalo * escalar
        anch_sep = self.__ancho_sepalo * escalar
        long_pet = self.__longitud_petalo * escalar
        anch_pet = self.__ancho_petalo * escalar

        return Flor(long_sep, anch_sep, long_pet, anch_pet, self.__especie)

# 3. Creo una clase abstracta para definir una interfaz de clasificación
class Clasificador(ABC):
    @abstractmethod
    def clasificar_por_ancho(self, flores):
        pass

# 4. Implemento la subclase con una estrategia específica de clasificación
class ClasificadorIris(Clasificador):
    def clasificar_por_ancho(self, flores):
        """Clasifico las flores según el ancho del sépalo en rangos definidos"""
        clasificacion = {
            "0.1-0.5": [],
            "0.6-1.0": [],
            "1.1-1.5": [],
            "1.6-2.0": [],
            "2.1-2.5": [],
            "2.6-3.0": []
        }

        for flor in flores:
            ancho = flor.ancho_sepalo
            if ancho <= 0.5:
                clasificacion["0.1-0.5"].append(flor)
            elif ancho <= 1.0:
                clasificacion["0.6-1.0"].append(flor)
            elif ancho <= 1.5:
                clasificacion["1.1-1.5"].append(flor)
            elif ancho <= 2.0:
                clasificacion["1.6-2.0"].append(flor)
            elif ancho <= 2.5:
```

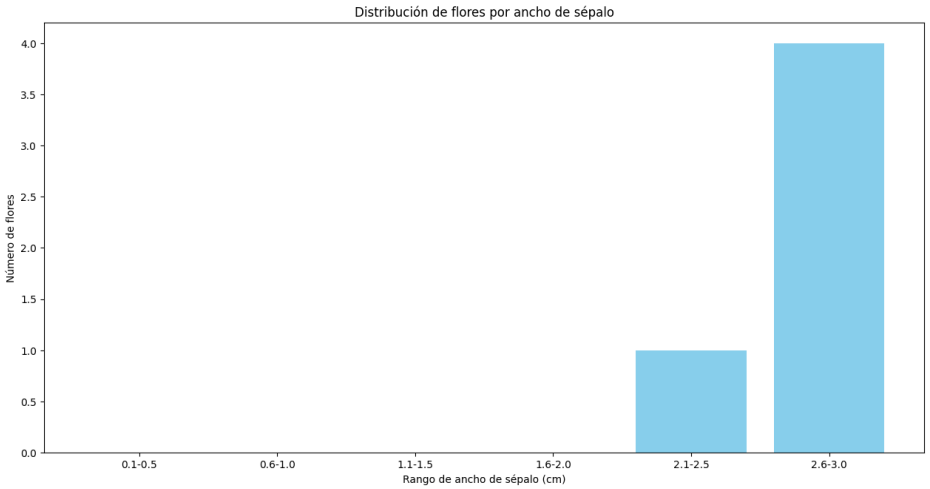
```
                clasificacion["2.1-2.5"].append(flor)
            else:
                clasificacion["2.6-3.0"].append(flor)

        # Devuelvo el número de flores en cada rango
        resultado = {rango: len(flores) for rango, flores in clasificacion.items()}
        return resultado

# 2. Instancio varias flores con datos del dataset Iris
flor1 = Flor(5.1, 3.5, 1.4, 0.2, "Iris-setosa")
flor2 = Flor(4.9, 3.0, 1.4, 0.2, "Iris-setosa")
flor3 = Flor(6.0, 2.7, 5.1, 1.6, "Iris-versicolor")
flor4 = Flor(6.3, 2.5, 5.0, 1.9, "Iris-virginica")
flor5 = Flor(5.8, 2.7, 5.1, 1.9, "Iris-virginica")

# Modifico algunos atributos para probar los setters
flor1.ancho_sepalo = 3.8
flor3.longitud_petalo = 4.9

# Muestro las flores
print("=== Flores creadas ===")
print(flor1)
print("\n" + str(flor2))
print("\n" + str(flor3))
print("\n" + str(flor4))
print("\n" + str(flor5))

# 5. Pruebo la sobrecarga del operador +
flor_promedio = flor1 + flor3
print("\n=== Flor promedio (flor1 + flor3) ===")
print(flor_promedio)

# Pruebo la sobrecarga del operador *
flor_escalada = flor2 * 1.5
print("\n=== Flor escalada (flor2 * 1.5) ===")
print(flor_escalada)

# 6. Comparo distancias entre flores
print("\n=== Comparación de distancias ===")
print(f"Distancia Manhattan entre flor1 y flor2: {flor1.distancia_manhattan(flor2):.2f}")
print(f"Distancia Chebyshev entre flor1 y flor2: {flor1.distancia_chebyshev(flor2):.2f}")

print(f"\nDistancia Manhattan entre flor3 y flor4: {flor3.distancia_manhattan(flor4):.2f}")
print(f"Distancia Chebyshev entre flor3 y flor4: {flor3.distancia_chebyshev(flor4):.2f}")
```

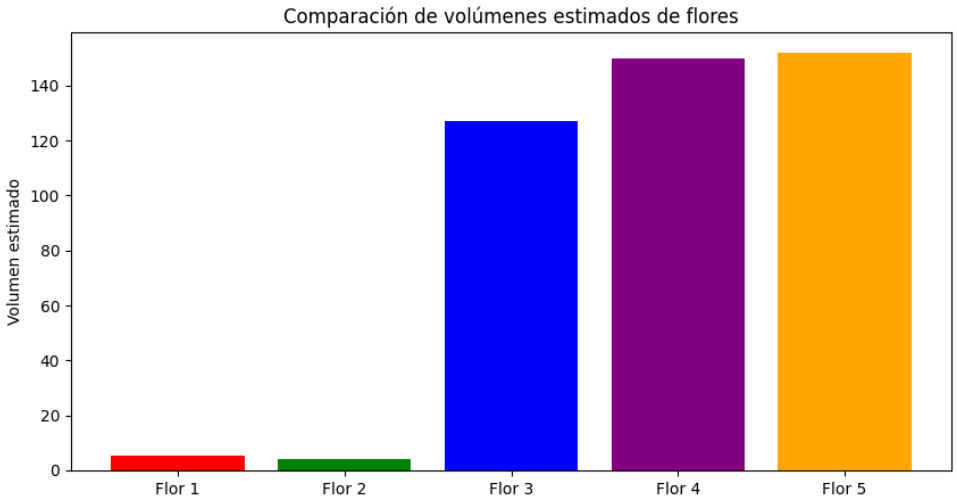
```
181.
182.# Busco cuál flor es la más cercana a flor1
183.distancias = {
184.    "flor2": flor1.distancia_manhattan(flor2),
185.    "flor3": flor1.distancia_manhattan(flor3),
186.    "flor4": flor1.distancia_manhattan(flor4),
187.    "flor5": flor1.distancia_manhattan(flor5)
188.}
189.
190.mas_cercana = min(distancias, key=distancias.get)
191.print(f"\nLa flor más cercana a flor1 es {mas_cercana} con distancia
    {distancias[mas_cercana]:.2f}")
192.
193.# 4. Clasifico flores según el ancho del sépalo
194.clasificador = ClasificadorIris()
195.flores = [flor1, flor2, flor3, flor4, flor5]
196.resultado_clasificacion = clasificador.clasificar_por_ancho(flores)
197.print("\n=== Clasificación por ancho de sépalo ===")
198.print(resultado_clasificacion)
199.
200.# 7. Visualizo los resultados con gráficos
201.# Gráfico de barras para la clasificación por ancho de sépalo
202.rangos = list(resultado_clasificacion.keys())
203.conteos = list(resultado_clasificacion.values())
204.
205.plt.figure(figsize=(10, 5))
206.plt.bar(rangos, conteos, color='skyblue')
207.plt.title("Distribución de flores por ancho de sépalo")
208.plt.xlabel("Rango de ancho de sépalo (cm)")
209.plt.ylabel("Número de flores")
210.plt.show()
211.
212.# Gráfico de barras para comparar volúmenes estimados
213.flores_nombres = ["Flor 1", "Flor 2", "Flor 3", "Flor 4", "Flor 5"]
214.volumenes = [flor.calcular_volumen_flor() for flor in flores]
215.
216.plt.figure(figsize=(10, 5))
217.plt.bar(flores_nombres, volumenes, color=['red', 'green', 'blue', 'purple',
    'orange'])
218.plt.title("Comparación de volúmenes estimados de flores")
219.plt.ylabel("Volumen estimado")
220.plt.show()
```

Prueba de escritorio

```
=== Flores creadas ===
Flor: Iris-setosa
Longitud sépalo: 5.1 cm
Ancho sépalo: 3.8 cm
Longitud pétalo: 1.4 cm
Ancho pétalo: 0.2 cm

Flor: Iris-setosa
Longitud sépalo: 4.9 cm
Ancho sépalo: 3.0 cm
Longitud pétalo: 1.4 cm
Ancho pétalo: 0.2 cm

Flor: Iris-versicolor
Longitud sépalo: 6.0 cm
Ancho sépalo: 2.7 cm
Longitud pétalo: 4.9 cm
Ancho pétalo: 1.6 cm

Flor: Iris-virginica
Longitud sépalo: 6.3 cm
Ancho sépalo: 2.5 cm
Longitud pétalo: 5.0 cm
Ancho pétalo: 1.9 cm

Flor: Iris-virginica
Longitud sépalo: 5.8 cm
Ancho sépalo: 2.7 cm
Longitud pétalo: 5.1 cm
Ancho pétalo: 1.9 cm

=== Flor promedio (flor1 + flor3) ===
Flor: Híbrido Iris-setosa-Iris-versicolor
Longitud sépalo: 5.55 cm
Ancho sépalo: 3.25 cm
Longitud pétalo: 3.1500000000000004 cm
Ancho pétalo: 0.9 cm

=== Flor escalada (flor2 * 1.5) ===
Flor: Iris-setosa
Longitud sépalo: 7.3500000000000005 cm
Ancho sépalo: 4.5 cm
Longitud pétalo: 2.0999999999999996 cm
Ancho pétalo: 0.30000000000000004 cm

=== Comparación de distancias ===
Distancia Manhattan entre flor1 y flor2: 1.00
Distancia Chebyshev entre flor1 y flor2: 0.80

Distancia Manhattan entre flor3 y flor4: 0.90
Distancia Chebyshev entre flor3 y flor4: 0.30

La flor más cercana a flor1 es flor2 con distancia 1.00

=== Clasificación por ancho de sépalo ===
{'0.1-0.5': 0, '0.6-1.0': 0, '1.1-1.5': 0, '1.6-2.0': 0, '2.1-2.5': 1, '2.6-3.0': 4}
```

Graficas


Distribución de flores por ancho de sépalo
Número de flores
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
0.1-0.5
0.6-1.0
1.1-1.5
1.6-2.0
2.1-2.5
2.6-3.0
Rango de ancho de sépalo (cm)


Comparación de volúmenes estimados de flores
Volumen estimado
0
20
40
60
80
100
120
140
Flor 1
Flor 2
Flor 3
Flor 4
Flor 5