import math

class Circle:

def \_\_init\_\_(self, radius=None, diameter=None):

"""

Initialize a Circle with either radius or diameter.

If diameter is provided, radius is calculated.

"""

if radius is not None:

self.radius = radius

elif diameter is not None:

self.radius = diameter / 2

else:

raise ValueError("You must provide either radius or diameter.")

@property

def diameter(self):

"""Return the diameter of the circle."""

return self.radius \* 2

def area(self):

"""Return the area of the circle."""

return math.pi \* self.radius \*\* 2

# Dunder method to print the circle

def \_\_str\_\_(self):

return f"Circle with radius: {self.radius:.2f}"

def \_\_repr\_\_(self):

return f"Circle(radius={self.radius:.2f})"

# Dunder method to add two circles

def \_\_add\_\_(self, other):

if isinstance(other, Circle):

return Circle(radius=self.radius + other.radius)

return NotImplemented

# Comparison dunder methods

def \_\_eq\_\_(self, other):

if isinstance(other, Circle):

return self.radius == other.radius

return NotImplemented

def \_\_lt\_\_(self, other):

if isinstance(other, Circle):

return self.radius < other.radius

return NotImplemented

def \_\_le\_\_(self, other):

if isinstance(other, Circle):

return self.radius <= other.radius

return NotImplemented

def \_\_gt\_\_(self, other):

if isinstance(other, Circle):

return self.radius > other.radius

return NotImplemented

def \_\_ge\_\_(self, other):

if isinstance(other, Circle):

return self.radius >= other.radius

return NotImplemented

# Example usage

c1 = Circle(radius=5)

c2 = Circle(diameter=10)

print(c1) # Circle with radius: 5.00

print(c2.diameter) # 10

print(f"Area of c1: {c1.area():.2f}") # Area of c1: 78.54

# Adding circles

c3 = c1 + c2

print(c3) # Circle with radius: 10.00

# Comparing circles

print(c1 > c2) # False

print(c1 < c2) # True

print(c1 == Circle(radius=5)) # True

# Sorting a list of circles

circle\_list = [c2, c1, c3]

circle\_list.sort()

print(circle\_list) # [Circle(radius=5.00), Circle(radius=5.00), Circle(radius=10.00)]

🔎 Explanation

Initialization (\_\_init\_\_):

You can create a circle using radius or diameter. Diameter is converted to radius automatically.

Properties:

diameter returns radius \* 2.

area() calculates the area using πr².

Printing (\_\_str\_\_ and \_\_repr\_\_):

Allows meaningful print outputs and representation.

Adding Circles (\_\_add\_\_):

Adding two Circle objects produces a new Circle with the sum of their radii.

Comparisons (\_\_eq\_\_, \_\_lt\_\_, \_\_le\_\_, \_\_gt\_\_, \_\_ge\_\_):

Compare radii of two Circle objects.

Enables usage in sort() and comparison operators (>, <, ==).

Sorting:

Circles can be sorted directly in a list because \_\_lt\_\_ is defined.

✅ This class covers all requested features: creation by radius/diameter, area calculation, printing, addition, comparison, and sorting.