מחלקות:

Circle:

from math import pi  
from Checkings import \*  
  
  
class Circle:  
 def \_\_init\_\_(self, radius: float):  
 *"""  
 Constructor that defines the circle's radius  
 :param radius: The value to define the circle radius value with  
 :raises AttributeError if the given value is not positive  
 """* self.radius = radius  
  
 def calculate\_area(self) -> float:  
 *"""  
 Method that calculates the circle's area  
 :return: float number that represents the circle's area  
 """* return pi \* (self.radius \*\* 2)  
  
 def calculate\_perimeter(self) -> float:  
 *"""  
 Method that calculates the circle's perimeter  
 :return: floating point value that represents the circle's perimeter  
 """* return 2 \* pi \* self.radius  
  
 def \_\_str\_\_(self) -> str:  
 *"""  
 Method that returns the radius in the specific format  
 :return: string that represents the circle's information  
 """* return f"Circle radius is:{self.radius}"  
  
 # Getters/setters  
 @property  
 def radius(self) -> float:  
 *"""  
 Getter that returns the circle's radius value  
 :return: floating point value that represents the circle's radius value  
 """* return self.\_\_radius  
  
 @radius.setter  
 def radius(self, radius: float) -> None:  
 *"""  
 Setter that defines the circle's radius  
 :param radius: A value to define the radius with  
 :raises AttributeError if the given value is not positive  
 :return: None  
 """* if not Checkings.is\_positive\_float(radius):  
 raise AttributeError("Radius must be a positive floating point value only")  
  
 self.\_\_radius = radius

main:

from Circle import \*  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 try:  
 c = Circle(3.5)  
  
 # Displaying circle's information, area and perimeter  
 print(c)  
 print(f"The area is :{round(c.calculate\_area(),2)}")  
 print(f"The perimeter is :{round(c.calculate\_perimeter(),2)}")  
 except AttributeError as a:  
 print(a)

Checking

class Checkings:  
 @staticmethod  
 def is\_positive\_float(value\_to\_check: float) -> bool:  
 *"""  
 Method checks if the given value is a positive floating point value  
 :param value\_to\_check: Value to validate  
 :return: True if the given value is a positive floating point value and False otherwise  
 """* return type(value\_to\_check) == float and

value\_to\_check > 0

test circle:

import unittest  
from Circle import Circle  
from math import pi  
  
  
class TestCircle(unittest.TestCase):  
 *"""  
 Unit test class for testing the Circle class.  
 """* \_\_test\_radius\_value = 3.0 # Default test radius value  
 \_\_test\_invalid\_values = [-1.0, 0.0, "string", None] # Invalid values for testing  
 \_\_test\_updated\_radius = 2.5 # Updated valid radius value  
 \_\_test\_initial\_radius = 1.5 # Initial valid radius value  
  
 def test\_valid\_radius(self):  
 *"""  
 Test that a valid radius is correctly assigned.  
 """* circle = Circle(self.\_\_test\_radius\_value)  
 self.assertEqual(circle.radius,

self.\_\_test\_radius\_value)  
  
 def test\_invalid\_radius(self):  
 *"""  
 Test that invalid radius values raise a AttributeError.  
 """* for value in self.\_\_test\_invalid\_values:  
 # Checking invalid radius values  
 with self.assertRaises(AttributeError):  
 Circle(value)  
  
 def test\_calculate\_area(self):  
 *"""  
 Test that the area calculation is correct.  
 """* circle = Circle(self.\_\_test\_radius\_value)  
 self.assertAlmostEqual(circle.calculate\_area(),

self.\_\_test\_radius\_value \*\* 2 \* pi)  
  
 def test\_calculate\_perimeter(self):  
 *"""  
 Test that the perimeter calculation is correct.  
 """* circle = Circle(self.\_\_test\_radius\_value)  
 self.assertAlmostEqual(circle.calculate\_perimeter(), 2 \* pi \* self.\_\_test\_radius\_value)  
  
 def test\_str\_method(self):  
 *"""  
 Test the \_\_str\_\_ method for correct output.  
 """* circle = Circle(self.\_\_test\_updated\_radius)  
 self.assertEqual(str(circle), f"Circle radius is:{self.\_\_test\_updated\_radius}")  
  
 def test\_setter\_valid\_radius(self):  
 *"""  
 Test that the radius setter correctly assigns a valid value.  
 """* circle = Circle(self.\_\_test\_initial\_radius)  
  
 # Updating the radius by using its setters  
 circle.radius = self.\_\_test\_updated\_radius  
  
 self.assertEqual(circle.radius, self.\_\_test\_updated\_radius)  
  
 def test\_setter\_invalid\_radius(self):  
 *"""  
 Test that assigning an invalid radius raises a AttributeError.  
 """* circle = Circle(self.\_\_test\_radius\_value)  
  
 for value in self.\_\_test\_invalid\_values:  
 # trying to update the radius value with wrong radius values  
 with self.assertRaises(AttributeError):  
 circle.radius = value  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 unittest.main()