Overview: In this HW, you will build on the code we developed together in class, as you learn more about inheritance in object-oriented programming (OOP).

Assignment: For this assignment, you will ultimately have five different source files, each containing a class definition corresponding to that filename:

- `Shape.py` (provided — do not modify)
- `Circle.py` (starter provided, which you must modify)
- `Triangle.py`
- `Rectangle.py`
- `Square.py`

Each of those files should have its own `main` function at the bottom, where testing for the specific class will occur. Remember to wrap your call to `main` using an `if __name__ == "__main__"` statement.

1. `Circle.py`: Complete the definition of the `Circle` class by completing the necessary methods:
   - `getRadius`
   - `setRadius`
   - `getPerimeter`
   - `__str__` (which will override the one inherited from `Shape`)
     - For this method, include "Circle" to start the string, and include the name, radius, and area.
     - Example output: `Circle sammy: radius = 3.0 area = 28.274333882308138`

   Use good style, including good naming conventions, and type hints for all parameters and function return type. (Because the methods here are self-explanatory, you may forego docstrings for this assignment only.)

   Include sufficient testing of all methods (including those inherited from `Shape`) inside the `main` within `Circle.py`. Keep in mind, for example, that if the user calls `setRadius` to change the radius of a circle, a subsequent call to the (inherited) `getArea` should return the correct updated area. As appropriate, clearly print what you are testing, what the actual result of your test is, and what the expected result of your test is. For example, the output of one of your tests might look like:

   ```
   radius = 4.0;  expected = math.pi * radius * radius
   c1 = Circle("sammy", radius)
   print(f"Testing c1.getRadius() -- result: {c1.getRadius()} expected: {expected}")
   ```

2. `Triangle.py`: Write a new class named `Triangle` that inherits from `Shape`. For this assignment, a triangle will be defined by its base and height, e.g., `t1 = Triangle("tommy", 4.0, 5.0)` where the name is passed to `__init__` in `Shape`, similar to that done in `Circle`.

   Then complete the necessary methods:

   - `getBase`, `getHeight`
   - `setBase`, `setHeight`
   - `__str__` (which will override the one inherited from `Shape`)
     - For this method, include "Triangle" to start the string, and include the name, base, height, and area.
     - Example output: `Triangle tommy: base = 4.0 height = 5.0 area = 10.0`
   - `__eq__` (which will override the one inherited from `Shape`)
     - Think carefully about why `__eq__` needs to be overridden inside `Triangle` but not in `Circle`.
     - Consider, for example, `Triangle("tommy", 4, 5)` versus `Triangle("tommy", 5, 4)`.

   Again, include sufficient testing of all methods (including those inherited from `Shape`) inside the `main` within `Triangle.py`. Keep in mind, for example, that if the user calls `setBase` to change the base of a triangle, a subsequent call to the (inherited) `getArea` should return the correct updated area; similar for `setHeight`.

3. Rectangle.py: Write a new class named `Rectangle` that inherits from `Shape`. For this assignment, a rectangle will be defined by its width and height. e.g., `r1 = Rectangle("robby", 4, 5)`.

   Then complete the necessary methods:

   - `getWidth`, `getHeight`
   - `setWidth`, `setHeight`
   - `getPerimeter`
   - `__str__` (which will override the one inherited from `Shape`)
     - For this method, include "Rectangle" to start the string, and include the name, width, height, and area.
     - Example output: `Rectangle robby: width = 4 height = 5 area = 20`
   - `__eq__` (which will override the one inherited from `Shape`)
     - Again, think carefully about why `__eq__` needs to be overridden here inside `Rectangle` but not in `Circle`.

   Again, include sufficient testing of all methods (including those inherited from `Shape`) inside the `main` within `Rectangle.py`. Keep in mind, for example, that if the user calls `setWidth` to change the width of a rectangle, a subsequent call to the (inherited) `getArea` should return the correct updated area; similar for `setHeight`.

4. Square.py: Write a new class named `Square` that inherits from `Rectangle` (**NOT** directly from `Shape`). For this assignment, a square will be defined by a single side length, e.g., `s1 = Square("sally", 5)`.

   Then complete the necessary methods:

   - `setWidth`, `setHeight`
     - Think carefully about why you should override these two methods inherited from `Rectangle`, but not `getWidth`, `getHeight`, `getArea`, or `getPerimeter`.
   - `__str__` (which will override the one inherited from `Rectangle`)
     - For this method, include "Square" to start the string, and include the name, side width, and area.
     - Example output: `Square sally: side = 5 area = 25`
   - Do you need to override the `__eq__` inherited from `Rectangle`? If so, include a new version here. Think carefully about whether you need to override the method.

   Again, include sufficient testing of all methods (including those inherited from `Rectangle` and `Shape`) inside the `main` within `Square.py`. Keep in mind, for example, that if the user calls `setWidth` to change the side length of a square, a subsequent call to the (inherited) `getArea` should return the correct updated area **AND** a call to the (inherited) `getHeight` or `getWidth` should return the correct updated same value; similar for `setHeight`.

Reflection: In the online text on Lyceum, provide meaningful responses to each of the following five prompts:

1. Why should there be a `getHeight` method in `Triangle` but not in `Shape`?
2. Why is there no `getPerimeter` method in `Triangle`?
3. Why is a new `__eq__` needed for `Rectangle` and `Triangle`, but not for `Circle` and not for `Square`? Be thorough.
4. Why do you need to override each of `setWidth` and `setHeight` inside `Square` and not just use the inherited versions from `Rectangle`?
5. Why would it have been better to have a `getArea` method only inside `Shape` (i.e., no corresponding `_area` instance variable), and instead just have subclasses override `getArea`? (Hint: consider the necessary `getWidth` and `getHeight` implementations for `Rectangle`.)