**Defining Properties Using Decorators in Python**

In Python, you can define properties in a class using the @property decorator. This allows you to control access to instance attributes by defining **getter**, **setter**, and **deleter** methods.

**Why Use Properties?**

* Encapsulation: Restrict direct access to private variables while allowing controlled access.
* Data Validation: Validate or modify values before assigning them.
* Computed Properties: Compute values dynamically rather than storing them as attributes.

**1. Using @property Decorator**

The @property decorator is used to define a getter method, which allows an attribute to be accessed like a regular attribute while still calling a method internally.

**Example: Read-Only Property**

class Circle:

def \_\_init\_\_(self, radius):

self.\_radius = radius # Private variable

@property

def radius(self):

"""Getter method for radius."""

return self.\_radius

# Usage

c = Circle(5)

print(c.radius) # 5

c.radius = 10 # AttributeError: can't set attribute

Here, radius behaves like an attribute but is actually a method.

**2. Defining a Setter Method**

To allow modification of the property, use @property along with a setter method using @property\_name.setter.

**Example: Read-Write Property**

class Circle:

def \_\_init\_\_(self, radius):

self.\_radius = radius

@property

def radius(self):

"""Getter method for radius."""

return self.\_radius

@radius.setter

def radius(self, value):

"""Setter method for radius."""

if value < 0:

raise ValueError("Radius cannot be negative")

self.\_radius = value

# Usage

c = Circle(5)

print(c.radius) # 5

c.radius = 10 # Works fine

print(c.radius) # 10

c.radius = -3 # Raises ValueError

Here, the setter ensures that the radius cannot be negative.

**3. Defining a Deleter Method**

To delete a property, use @property\_name.deleter.

**Example: Deleting a Property**

class Circle:

def \_\_init\_\_(self, radius):

self.\_radius = radius

@property

def radius(self):

return self.\_radius

@radius.setter

def radius(self, value):

if value < 0:

raise ValueError("Radius cannot be negative")

self.\_radius = value

@radius.deleter

def radius(self):

print("Deleting radius")

del self.\_radius

# Usage

c = Circle(5)

del c.radius # Prints: Deleting radius

After deletion, trying to access c.radius will raise an AttributeError.

**4. Computed Properties**

A property does not need to store a value—it can compute it dynamically.

**Example: Calculating Area of a Circle**

import math

class Circle:

def \_\_init\_\_(self, radius):

self.\_radius = radius

@property

def area(self):

"""Computes area dynamically"""

return math.pi \* (self.\_radius \*\* 2)

c = Circle(5)

print(c.area) # 78.53981633974483

c.area = 100 # AttributeError: can't set attribute

The area property is computed dynamically based on the radius.

**Summary**

| **Decorator** | **Purpose** |
| --- | --- |
| @property | Defines a getter method |
| @property\_name.setter | Defines a setter method |
| @property\_name.deleter | Defines a deleter method |

Properties allow controlled access to attributes while enabling encapsulation and validation.