1. Overview of Classes and Methods

- A class is a blueprint; instances (objects) are created based on this blueprint.
- Example: If you design a blueprint for a house, you can build many houses (instances) from it.
- Different instances may have different attributes (e.g., color).
- To modify or interact with these attributes, methods are used inside the class.

2. Instance Methods

- Instance methods operate on individual instances of a class.
- Defined with def inside a class, their first parameter is always self, referring to the instance.

Example given:

```
class House:
    def __init__(self, color):
        # 'self.color' stores the paint color for this specific house instance
        self.color = color

    def show_color(self):
        # Prints the color of this house instance using 'self'
        print(f"This house is painted {self.color}")

# Create an instance of House with color "blue"
my_house = House("blue")

# Call the method to display the house color
my_house.show_color()
```

Output:

```
This house is painted red
```

- The keyword self keeps track of the specific instance and its attributes.
- Instance methods handle changes at the instance level.

Instance Methods = Methods Bound to an Instance
Instance methods are functions defined inside a class that operate on an instance of the class. They take `self` as their first parameter, which refers to the specific instance

calling the method. These methods can access and modify the attributes of the instance and call other instance methods.

```
class House:
    # Constructor to initialize the house with a specific color
    def __init__(self, color):
        self.color = color  # Assign the color to the house instance

    # Instance method to display the color of the house
    def show_color(self):
        print(f"This house is painted {self.color}")  # Access and print the house's color

# Create an instance of the House class with the color "red"
my_house = House("red")

# Call the instance method to show the color of the house
my_house.show_color()
```

3. Class Methods

```
class House:
    def __init__(self, color, age):
        self.color = color      # Instance attribute: specific color of this house
        self.age = age          # Instance attribute: how old the house is

    @classmethod
    def from_construction_year(cls, color, construction_year):
        # Class method to create a House object by calculating age from construction year
        return cls(color, 2024 - construction_year)

# Create a new house instance using the class method as a factory
my_house = House.from_construction_year("white", 2000)

# Print the age of the house instance
print(my_house.age)
```

- Class methods are linked to the class itself, not individual instances.
- Decorated with @classmethod.
- Their first parameter is cls, which refers to the class.

- They interact with or manipulate class-level attributes.
- Example:
    - Define the class method with decorator @classmethod.
    - Call using ClassName.methodName() passing relevant parameters.
- Example from notebook:
    - A method calculates construction year or age using class attributes.
- Instances can be created using class methods as alternate constructors.

Example of calling:
House.calculate_age(2025)

You access attributes of the instance using the object:
myHouse.age

4. Static Methods

- A static method is a method that belongs to a class but does not access or modify the class itself or any instance of the class. Unlike instance methods or class methods, static methods do not take self (the instance) or cls (the class) as their first parameter.
- Static methods behave like regular functions, but they are logically grouped inside a class because they perform operations relevant to the class's purpose.
- They are typically used for utility tasks such as validations, calculations, or helper functions that don't rely on the state of the class or its instances.
- Static methods are defined using the @staticmethod decorator.

```
class House:
    def __init__(self, color, area):
        self.color = color
        self.area = area  # Area in square meters

    @staticmethod
    def is_valid_area(area):
        # Static method to validate if the area value is reasonable (positive)
        return area > 0
```

```
# Using the static method to validate area before creating a House instance
print(House.is_valid_area(120))   # Expected output: True
print(House.is_valid_area(-50))   # Expected output: False
```

- Static methods are independent utility functions inside a class.
- Decorated with @staticmethod.
- They do not take self or cls as parameters.
- Used as helpers or validators related to the class but not dependent on class or instance data.
- Example:
    - A method isValidArea validates if a given area is positive or not.
    - Returns True if valid, False otherwise.
- They contribute as building blocks to larger class functionalities.

5. Difference Between Functions and Methods

- Functions are defined outside classes using def.
- Methods are functions defined inside classes and categorized as instance, class, or static methods depending on their first parameter and usage.
- When a function is placed inside a class, it becomes a method.

6. Combined Example: MathTools Class

- Class MathTools contains:
    - A constant value PI.
    - A constructor taking radius as an attribute.
    - An instance method area that calculates area using radius.
    - A class method diameter calculating diameter using class method decorator.
    - A static method isValidArea that validates the input area.
- Usage in sequence:
    - Create an object and call instance method area.

        - Call class method diameter via class name.
        - Call static method isValidArea via class name.
    - The notebook contains detailed examples and challenges for self-practice.

Story-Driven Example

```
class MathTool:
    pi = 3.14159
    def __init__(self, radius):
        self.radius = radius
    # Instance method to calculate the area of a circle
    def area(self):
        return MathTool.pi * self.radius * self.radius
    # Class method to create a MathTool object from diameter
    @classmethod
    def from_diameter(cls, diameter):
        return cls(diameter / 2)
    # Static method to validate if a radius is positive
    @staticmethod
    def is_valid_radius(radius):
        return radius > 0
# Example usage:
tool = MathTool(5)
print(tool.area())  # Output: 78.53975
tool_from_diameter = MathTool.from_diameter(10)
print(tool_from_diameter.radius)  # Output: 5.0
print(MathTool.is_valid_radius(-1))  # Output: False
```

7. Summary

- Understanding methods in OOP is critical to manipulate objects effectively.
- Rigorous hands-on practice with examples, such as those in the provided notebook, will deepen comprehension.
- Studying code line-by-line using AI tools (ChatGPT, Claude, etc.) is encouraged for thorough understanding.
- A comparison table summarizing the differences and calling conventions of instance, class, and static methods is available in the notebook for easy reference.