**Static Method in Python**

A **static method** in Python is a method that belongs to a class rather than an instance of the class. Unlike instance methods, static methods do not require access to the instance (i.e., self) or class-level data (i.e., cls). Instead, they are general utility functions that can be called on a class directly and are not concerned with class or instance state.

Static methods are defined using the @staticmethod decorator in Python. These methods can be called either on the class itself or on an instance of the class.

**Key Points about Static Methods:**

* Static methods do not modify class or instance state.
* They don’t require access to instance-specific data (self).
* They can be called on the class without creating an instance.

**Example of a Static Method**

python

Copy code

class MathOperations:

# Static method to calculate the square of a number

@staticmethod

def square(number):

return number \* number

# Static method to add two numbers

@staticmethod

def add(a, b):

return a + b

# Call static methods without creating an instance

print(MathOperations.square(5)) # Output: 25

print(MathOperations.add(10, 20)) # Output: 30

# You can also call them from an instance (but it's not necessary)

math\_ops = MathOperations()

print(math\_ops.square(6)) # Output: 36

**Explanation:**

1. **Defining Static Methods**:
   * The @staticmethod decorator is used to define the static methods square() and add() within the MathOperations class.
   * These methods do not access any instance or class-level data.
2. **Calling Static Methods**:
   * You can call MathOperations.square(5) or MathOperations.add(10, 20) directly without creating an instance of the class.
   * Although it is possible to call static methods using an instance of the class (e.g., math\_ops.square(6)), it is not necessary and usually not the intended use.

**When to Use Static Methods:**

* **Utility functions**: If a method performs an operation that is related to the class but does not require access to class-specific or instance-specific data, it is a good candidate for being a static method.
* **Organizing code**: Static methods allow you to group related functions within a class, even if those functions don’t need to access class or instance data.

**Example: Practical Use Case of Static Methods**

Consider a class that validates user inputs. This validation logic doesn't need access to any instance attributes, so it can be defined as a static method.

python

Copy code

class Validator:

# Static method to check if a string is alphanumeric

@staticmethod

def is\_alphanumeric(input\_string):

return input\_string.isalnum()

# Static method to check if an email is valid (simple validation)

@staticmethod

def is\_valid\_email(email):

return "@" in email and "." in email

# Using static methods without creating an instance

print(Validator.is\_alphanumeric("Hello123")) # Output: True

print(Validator.is\_valid\_email("test@example.com")) # Output: True

print(Validator.is\_valid\_email("testexample")) # Output: False

In this example, the Validator class contains static methods to validate strings, such as checking if a string is alphanumeric or if an email is valid. These methods don't need to know anything about specific instances, so they are static.

**Difference Between Static Methods and Class Methods:**

* **Static Methods**: They don’t require a reference to the class (cls) or instance (self). They act like regular functions but belong to the class’s namespace.
* **Class Methods**: They take a reference to the class (cls) as the first parameter and can modify class state. Class methods are defined using the @classmethod decorator.

**Conclusion:**

* **Static methods** are general utility methods that belong to the class but do not require access to instance or class-level data.
* They are defined using the @staticmethod decorator and are useful when you need a method to perform a task that is logically connected to a class but does not modify its state.