Day 38

Static Methods in Python:

What is a Static Method?

A static method is a method that:

- Belongs to a class, not to any specific object (instance)
- Does not use:
  - self (instance data)
  - cls (class data)
- Behaves like a regular function, but lives inside a class for logical grouping

Think of it as:

"A utility function related to a class"

Why Do We Need Static Methods?

Static methods are used when:

- The logic conceptually belongs to a class
- But it doesn't need object data or class data

Example idea: A class MathUtils that groups math-related functions.

Normal Method vs Static Method

Normal (Instance) Method:

```
class Example:
  def instance_method(self):
    print("Needs an object")
```

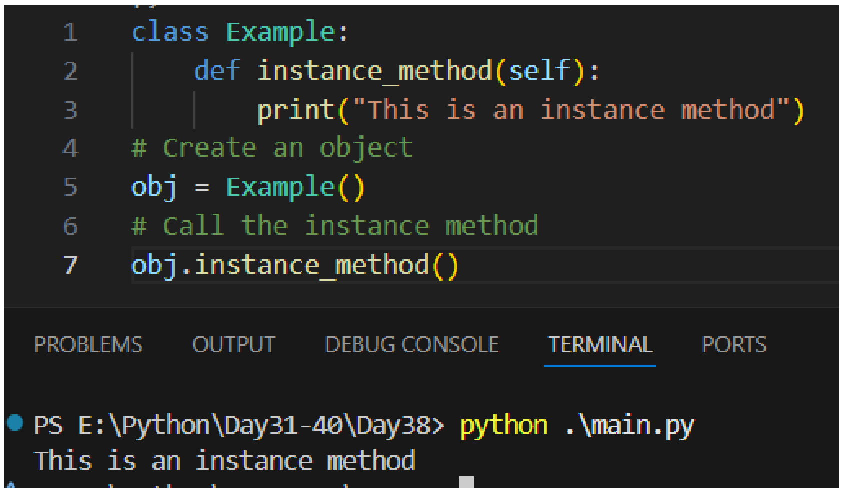
- Requires an object
- Uses self

Static Method:

```
class Example:
  @staticmethod
  def static_method():
    print("Does NOT need an object")
```

- No self
- No cls
- Can be called without creating an object

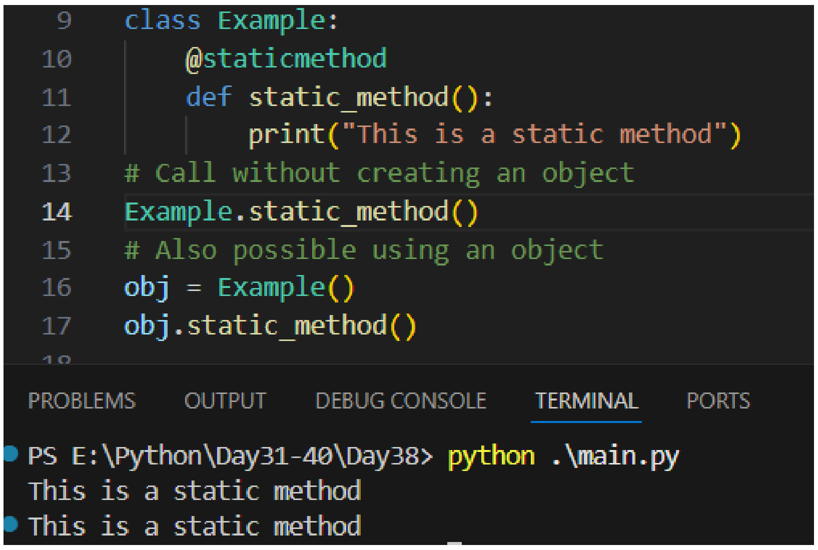
Example: normal (instance) method.



```python
class Example:
    def instance_method(self):
        print("This is an instance method")
# Create an object
obj = Example()
# Call the instance method
obj.instance_method()
```

```
PS E:\Python\Day31-40\Day38> python .\main.py
This is an instance method
```

Example: static method.



```python
class Example:
    @staticmethod
    def static_method():
        print("This is a static method")
# Call without creating an object
Example.static_method()
# Also possible using an object
obj = Example()
obj.static_method()
```

```
PS E:\Python\Day31-40\Day38> python .\main.py
This is a static method
This is a static method
```

Summary:

- Are defined using @staticmethod
- Do not use self or cls
- Are called using the class name
- Help organize related functions

Instance variables vs Class variables in Python:

What Are Variables in a Class?

In Python, variables inside a class are of two types:

1. Instance Variables → Belong to an object
2. Class Variables → Belong to the class itself

Instance Variables (Object Variables)

Instance variables:

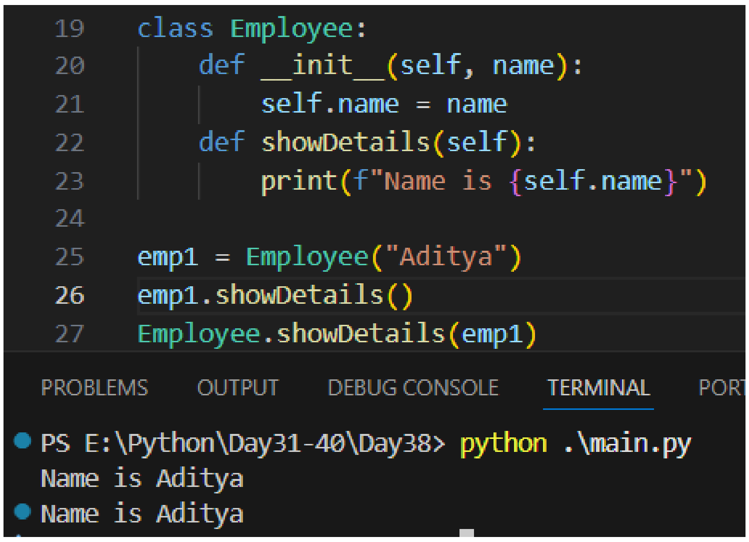
- Are unique to each object
- Are created using self
- Store data that differs from object to object

Class Variables (Static Variables)

Class variables:

- Are shared by all objects
- Are declared inside the class but outside methods
- Store data common to all objects

Example: a very important example before anything. Clearly, both "emp1.showDetails()" and "Employee.showDetails(emp1)" are equivalent to each other, as the output for both is same.

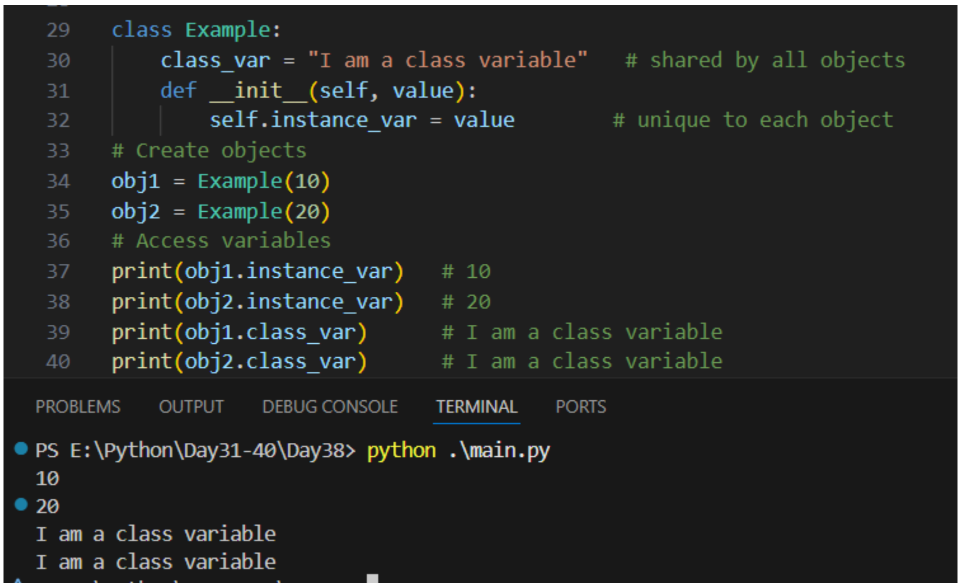


```python
class Employee:
    def __init__(self, name):
        self.name = name
    def showDetails(self):
        print(f"Name is {self.name}")

emp1 = Employee("Aditya")
emp1.showDetails()
Employee.showDetails(emp1)
```

```
PS E:\Python\Day31-40\Day38> python .\main.py
Name is Aditya
Name is Aditya
```

Example:



```python
class Example:
    class_var = "I am a class variable"   # shared by all objects
    def __init__(self, value):
        self.instance_var = value        # unique to each object
# Create objects
obj1 = Example(10)
obj2 = Example(20)
# Access variables
print(obj1.instance_var)   # 10
print(obj2.instance_var)   # 20
print(obj1.class_var)      # I am a class variable
print(obj2.class_var)      # I am a class variable
```

```
PS E:\Python\Day31-40\Day38> python .\main.py
10
20
I am a class variable
I am a class variable
```

Summary:

➔ **Instance Variables**

- Belong to an object
- Created using self.variable
- Defined inside methods (usually __init__)
- Separate copy for each object
- Change affects only that object
- Used for object-specific data

➔ **Class Variables**

- Belong to the class
- Defined inside the class, outside methods
- Single shared copy for all objects
- Change affects all objects
- Used for common/shared data

➔ **Key Point**

- Changing a class variable using an object name creates an instance variable

--The End--