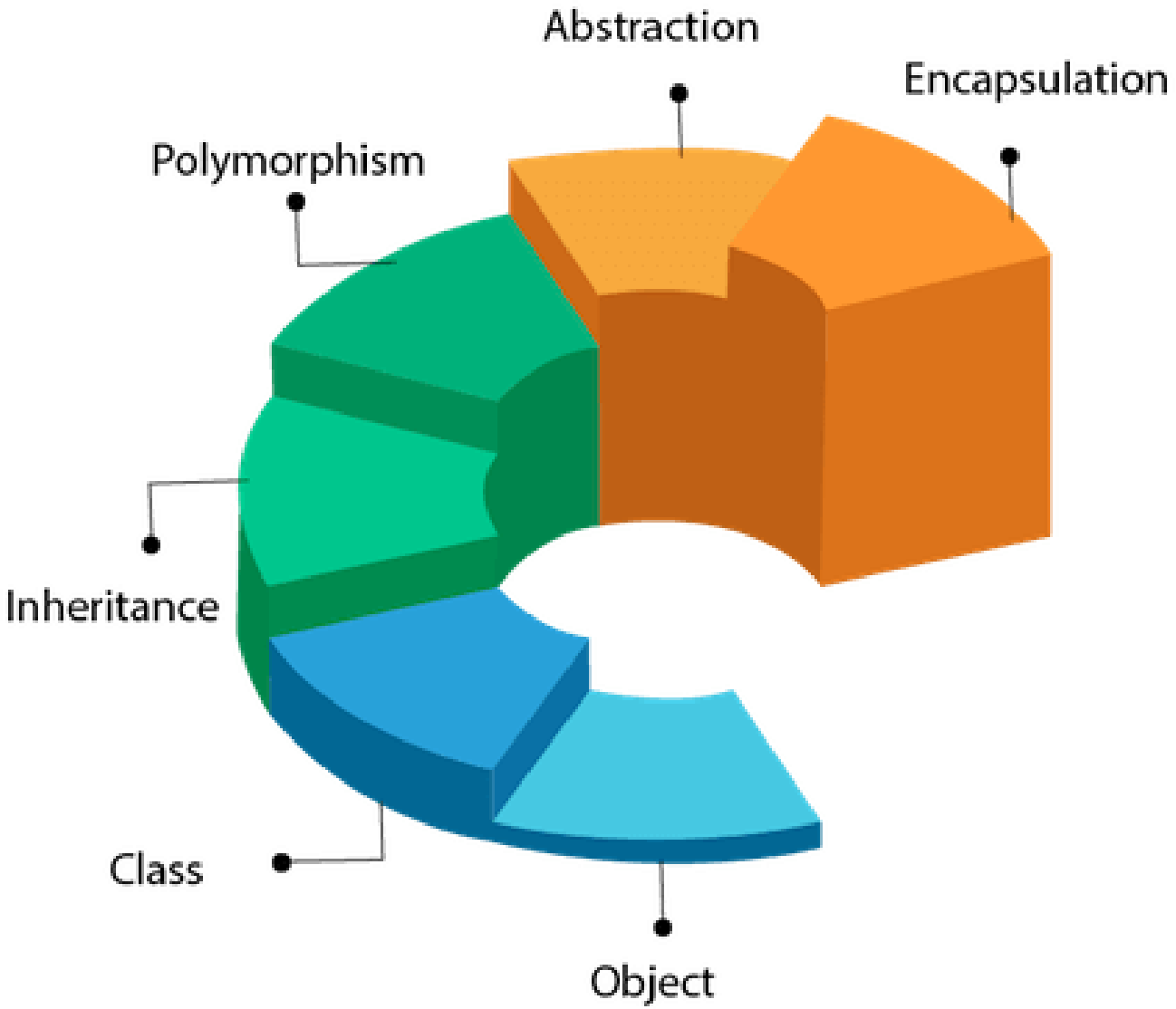
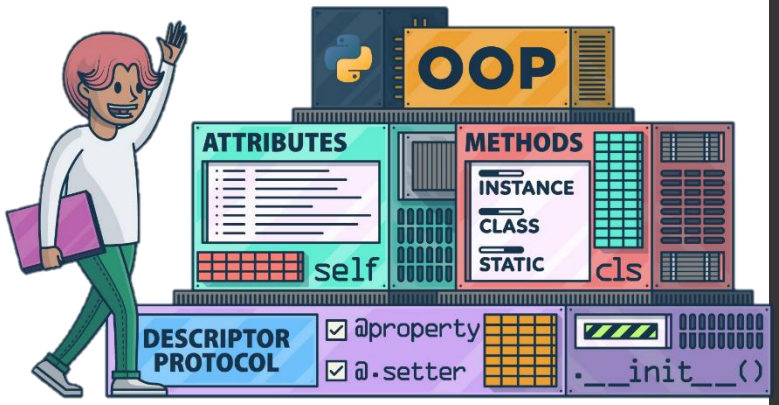
Object-Oriented Programming (OOP)

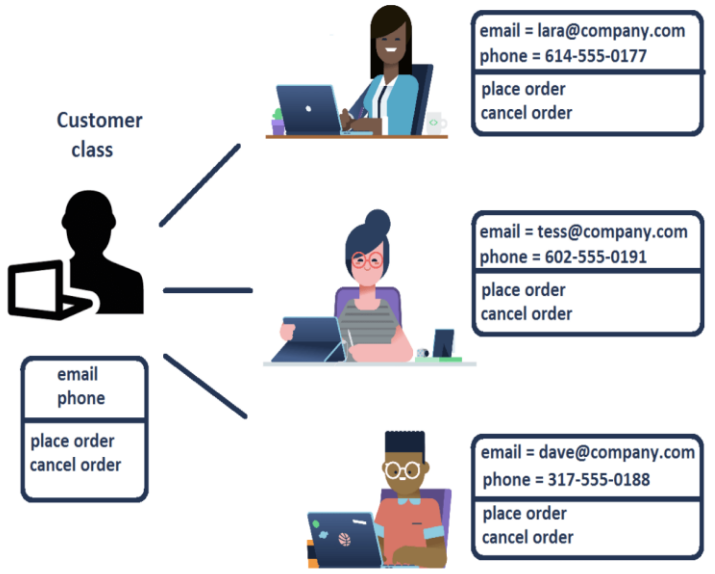
OOPs (Object-Oriented Programming System)



Classes & Objects

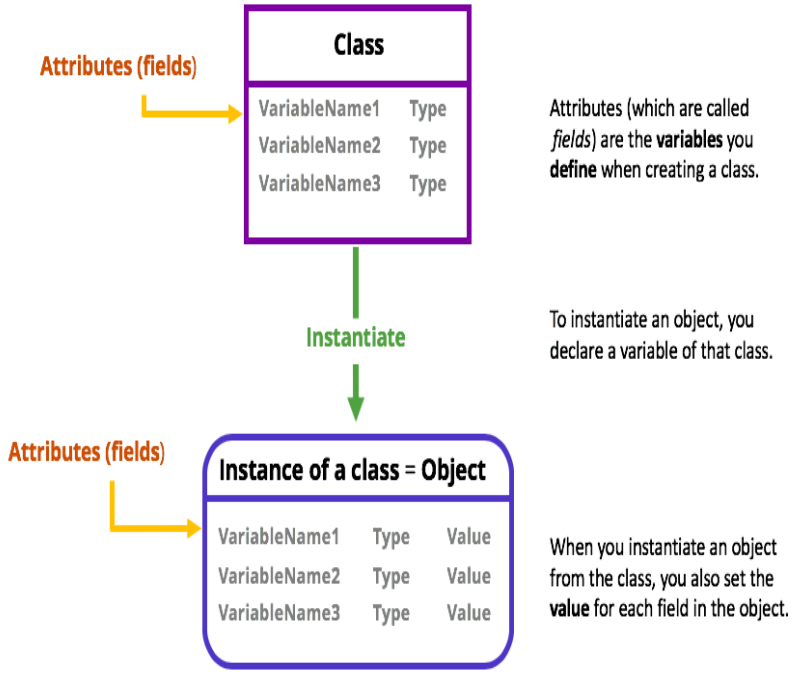
- **Header:** Classes & Objects in Python
- Classes are blueprints or templates for creating objects. Think of them as a real-world entity.
- Objects are instances of classes, representing a specific example of the class.
- Objects can store data (attributes) and have behavior (methods).
- **Example:**
  - class Dog:
  - pass
  - my_dog = Dog()
- **Key Concept:** Every object created from a class can have different attribute values, but all share the same structure.
- **Benefits:**
  - **Reusability:** Once a class is defined, you can create multiple objects from it.
  - **Modularity:** Encapsulation of data and methods within a class.

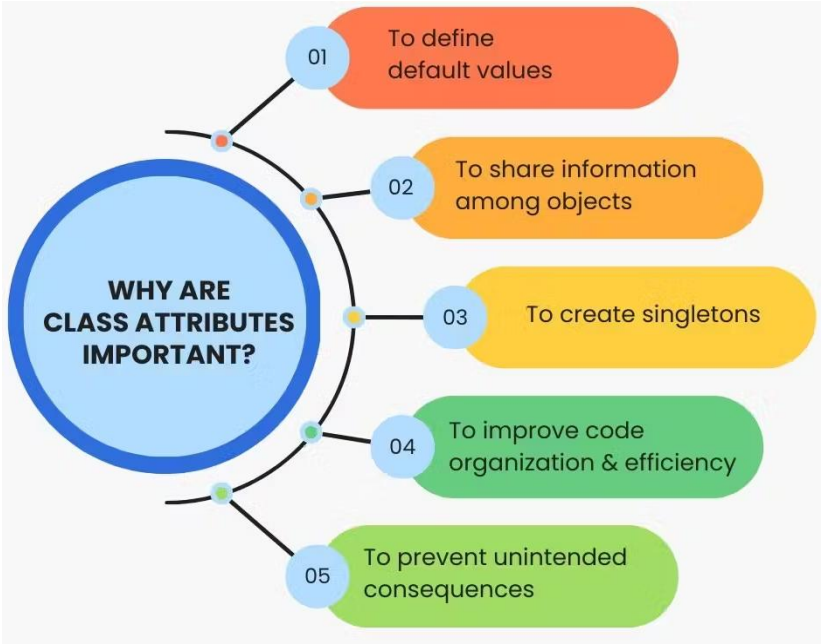




Attributes and Methods

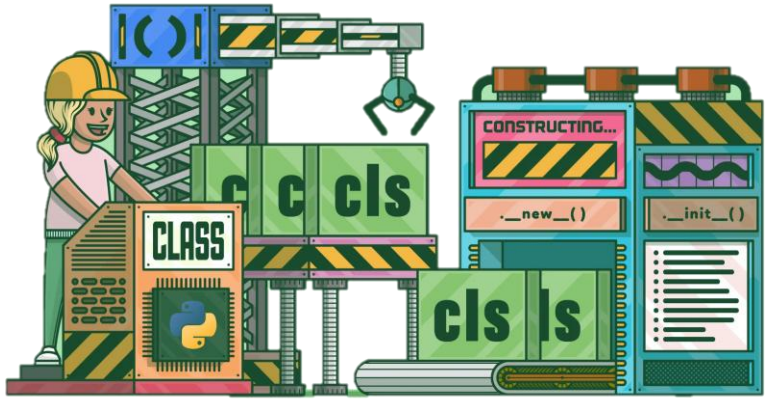
- **Header:** Attributes and Methods
- Attributes (Fields/Properties): Variables that store the state of an object.
- **Instance Attributes:** Data unique to each object.
- **Methods:** Functions defined inside a class that describe the behavior of the object.
- **Example:**
- class Dog:
- def __init__(self, name, breed):
- self.name = name # Attribute
- self.breed = breed # Attribute
- def bark(self): # Method
- print(f"{self.name} is barking!")
- **Methods' Role**: Operate on the instance's data and can change the object's state.
- **Attributes' Role:** Represent the object's characteristics (e.g., name, breed, age).

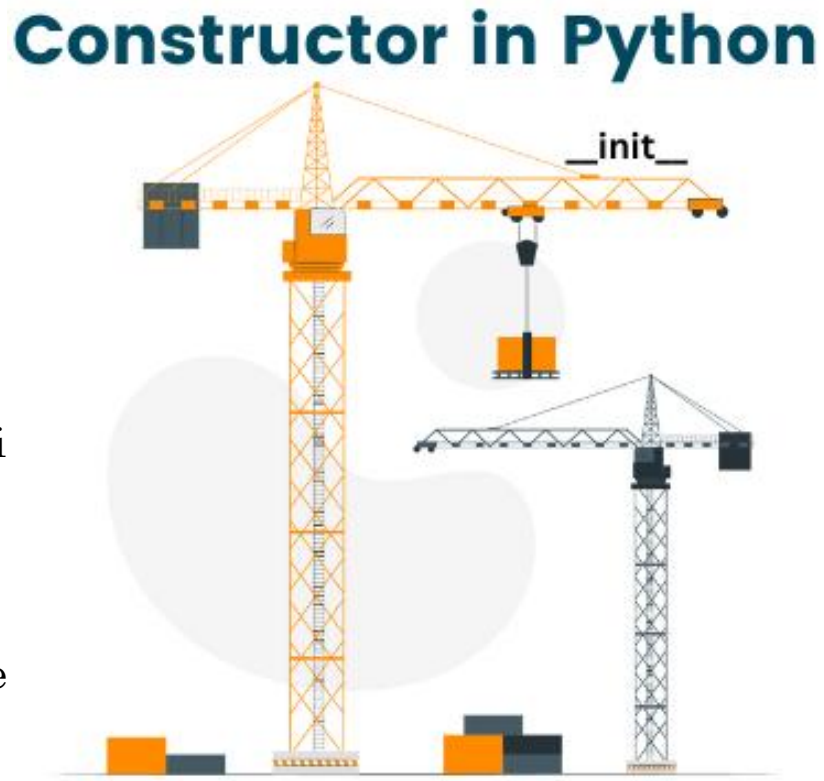




Understanding the __init__ Constructor Method

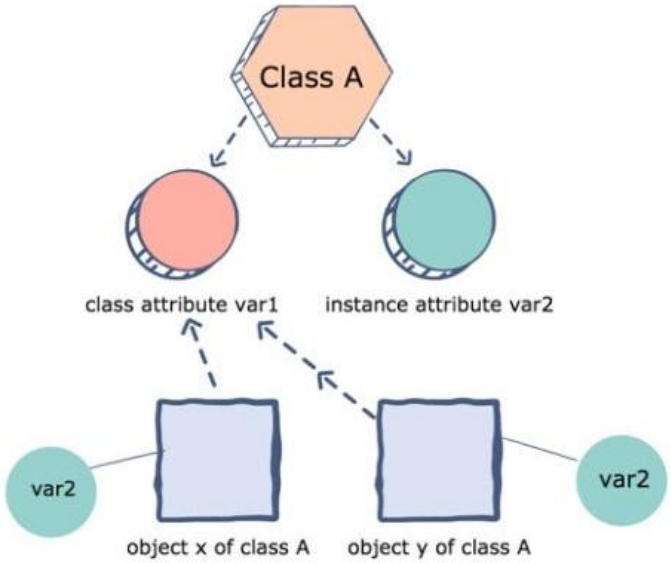
- **Header:** The __init__ Method (Constructor)
- **Definition:** Special method automatically called when a new object of the class is created.
- **Purpose:** Used to initialize the object's attributes.
- **self keyword:** Refers to the instance of the class and allows access to its attributes and methods.
- **Example**:
- class Dog:
- def __init__ (self, name, age):
- self.name = name
- self.age = age
- **Explanation:** The __init__ method sets the initial values for attributes li name and age upon object creation.
- **Key Points:**
- self is mandatory in method definitions inside the class to access instance variables.
- Can be used to validate or set default values for attributes





Class Attributes vs Instance Attributes

- **Header:** Class Attributes vs Instance Attributes
- Instance Attributes: Specific to a particular instance/object, defined within __init__.
- Class Attributes: Shared across all instances, defined directly within the class.
- **Example:**
  - class Dog:
  - species = 'Canine' # Class Attribute
  - def __init__ (self, name, age):
  - self.name = name # Instance Attribute
  - self.age = age
- **Explanation:**
- Class attributes are the same for all objects created from that class.
- Instance attributes vary from object to object.
- **Use Case:** Class attributes for data common to all objects (e.g., all dogs are of spe 'Canine').
- **Visual Representation:**
  - Dog 1: name = 'Rex', age = 2.
  - Dog 2: name = 'Buddy', age = 4.
  - Both have species = 'Canine' as a shared class attribute.



Class attribute defined at top of class

```
>>> class Person:
...     company = "ucd"
...
...     def __init__(self):
...         self.age = 23
```

Instance attribute defined inside a class function. The self prefix is always required.

```
>>> p1 = Person()
>>> p2 = Person()
>>> p1.age = 35
>>> print p2.age
23
```

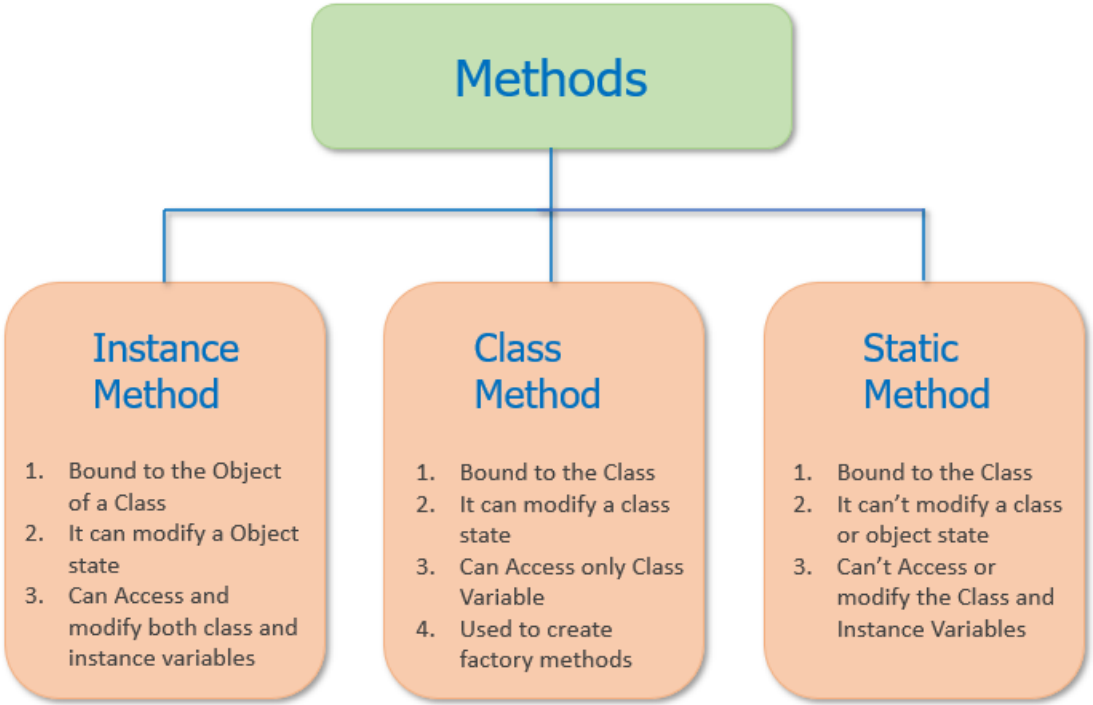
Change to instance attribute age affects only the associated instance (p2)

```
>>> p1 = Person()
>>> p2 = Person()
>>> p1.company = "ibm"
>>> print p2.company
'ibm'
```

Change to class attribute company affects all instances (p1 and p2)

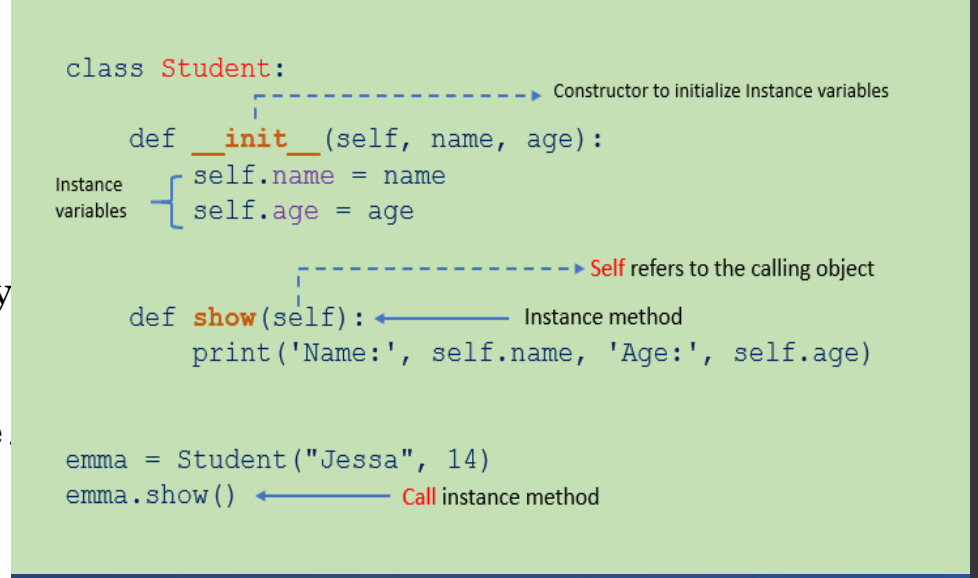
Types of Methods

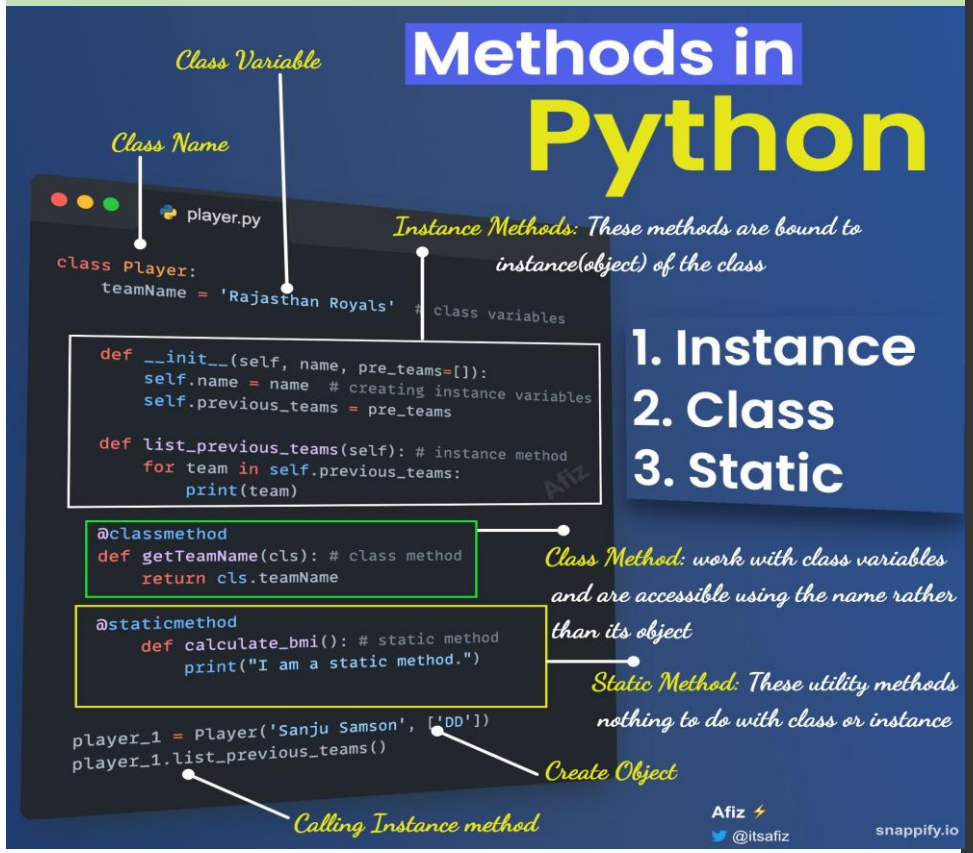
- **Header:** Different Types of Methods
- **Instance Methods:** Operate on an instance of the class and access/modify instance attributes.
- **Class Methods:** Operate on the class itself and have access to class attributes.
- **Static Methods:** Standalone functions within a class. Don't operate on instances or the class itself.
- **Use Cases:**
- Instance methods for object-specific behavior.
- Class methods for operations related to the class as a whole (e.g., factory methods).
- Static methods for utility/helper functions.



Instance Methods

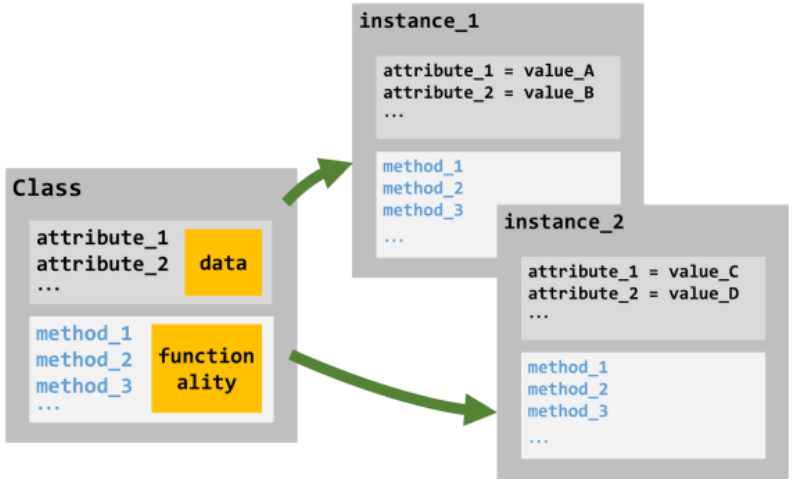
- **Header:** Instance Methods
- **Definition:** Operate on individual objects of the class and typically modify instance attributes.
- Requires self as the first parameter to refer to the specific instance
- **Example:**
- class Dog:
- def __init__ (self, name):
- self.name = name
- def bark(self):
- print(f"{self.name} barks!")
- **Explanation:** self allows each instance to access its own data.
- **Key Points:**
- Used to manipulate or retrieve instance-specific data.
- Can modify object state.





Class Methods

- Definition: Methods that operate on the class itself and typically use @classmethod decorator.
- Uses cls as the first parameter to refer to the class.
- Example:
- class Dog:
- species = 'Canine'
- @classmethod
- def show_species(cls):
- print(f"All dogs are {cls.species}.")
- Explanation: Class methods are often used for operations that affect the class as a whole, like modifying class-level attributes.
- Use Case: Factory methods that create instances in different ways or manage shared class-level behavior.



Syntax of a Python Class:

[ ]:

```
class ClassName:
    # Class variables
    class_variable = value

    # Constructor
    def __init__(self, parameters):
        self.instance_variable = parameters

    # Instance method
    def method_name(self, parameters):
        # method body

#clcoding.com
```

Static Methods

- Header: Static Methods
- Definition: A static method does not depend on instance or class; behaves like a regular function, but it is inside a class.
- Uses the @staticmethod decorator.
- Example:
- class Dog:
- @staticmethod
- def info():
- print("Dogs are loyal animals.")
- **Explanation**: Static methods are used for utility functions within the class that don't need access to class or instance attributes.
- **Key Points:**
- No self or cls arguments required.
- Useful for grouping logically related functions with the class, even if they don't modify object or class state.