**1. Class and Object Basics**

**Command: class**

Defines a class.

-

-

class Animal:

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

self.name = name # Attribute

**Command: Creating an Object**

Instantiating a class to create an object.

-

-

dog = Animal("Dog")

print(dog.name) # Output: Dog

**2. Inheritance**

**Command: Class Inheritance**

A child class inherits from a parent class.

-

-

class Dog(Animal):

def speak(self):

return f"{self.name} says Woof!"

**Command: Using Parent Class**

Access methods or attributes of a parent class.

-

-

dog = Dog("Buddy")

print(dog.speak()) # Output: Buddy says Woof!

**3. Overriding Methods**

**Command: Overriding Parent Class Methods**

Redefining a method in the child class.

-

-

class Cat(Animal):

def speak(self):

return f"{self.name} says Meow!"

-

-

cat = Cat("Kitty")

print(cat.speak()) # Output: Kitty says Meow!

**4. Polymorphism**

**Command: Method Overloading (via Default Arguments)**

- doesn’t support traditional method overloading but achieves similar functionality with default arguments.

-

-

class Calculator:

def add(self, a, b=0):

return a + b

-

-

calc = Calculator()

print(calc.add(5)) # Output: 5

print(calc.add(5, 10)) # Output: 15

**5. Special (Dunder) Methods**

**Command: \_\_init\_\_**

Constructor method.

-

-

class Person:

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

**Command: \_\_str\_\_**

String representation of the object.

-

-

class Person:

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

def \_\_str\_\_(self):

return f"{self.name}, {self.age} years old"

-

-

person = Person("Alice", 30)

print(person) # Output: Alice, 30 years old

**Command: \_\_repr\_\_**

Formal string representation.

-

-

class Person:

def \_\_repr\_\_(self):

return f"Person(name={self.name}, age={self.age})"

**Command: \_\_eq\_\_**

Defines equality between two objects.

-

-

class Point:

def \_\_init\_\_(self, x, y):

self.x = x

self.y = y

def \_\_eq\_\_(self, other):

return self.x == other.x and self.y == other.y

-

-

p1 = Point(1, 2)

p2 = Point(1, 2)

print(p1 == p2) # Output: True

**6. Access Modifiers**

**Command: Public Attribute**

Default behavior (accessible everywhere).

-

-

class Animal:

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

self.name = name # Public

**Command: Protected Attribute**

Prefix with \_ to indicate "protected".

-

-

class Animal:

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

self.\_species = name # Protected

**Command: Private Attribute**

Prefix with \_\_ to make the attribute private.

-

-

class Animal:

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

self.\_\_secret = "This is hidden!" # Private

-

-

animal = Animal("Tiger")

# print(animal.\_\_secret) # Error: AttributeError

**7. Property Decorators**

**Command: @property**

Used to define getter methods.

-

-

class Circle:

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

self.\_radius = radius

@property

def radius(self):

return self.\_radius

**Command: @<property>.setter**

Define setter methods.

-

-

@radius.setter

def radius(self, value):

if value < 0:

raise ValueError("Radius cannot be negative!")

self.\_radius = value

**8. Static and Class Methods**

**Command: @staticmethod**

Method that doesn’t access class or instance variables.

-

-

class Math:

@staticmethod

def add(a, b):

return a + b

-

-

print(Math.add(3, 5)) # Output: 8

**Command: @classmethod**

Method that operates on the class itself.

-

-

class Person:

count = 0

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

self.name = name

Person.count += 1

@classmethod

def total(cls):

return cls.count

-

-

print(Person.total()) # Output: 0

p1 = Person("Alice")

print(Person.total()) # Output: 1

**9. Abstract Classes**

**Command: @abstractmethod**

Defines methods that must be implemented by subclasses.

-

-

from abc import ABC, abstractmethod

class Shape(ABC):

@abstractmethod

def area(self):

pass

-

-

class Circle(Shape):

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

self.radius = radius

def area(self):

return 3.14 \* self.radius\*\*2

**10. Multiple Inheritance**

**Command: Inheriting from Multiple Classes**

-

-

class A:

def method\_a(self):

return "A"

class B:

def method\_b(self):

return "B"

class C(A, B):

pass

c = C()

print(c.method\_a(), c.method\_b()) # Output: A B