Chapter 2 - Classes and Objects

1. Introduction to Classes and Objects

Object-Oriented Programming (OOP) is a paradigm that helps structure programs into reusable and modular code. The fundamental building blocks of OOP are **classes** and **objects**.

1.1 What is a Class?

A **class** is a blueprint for creating objects. It defines attributes (data) and methods (functions) that describe the behavior of the objects.

Syntax of a Class

```
class Car:
    def __init__(self, brand, model, year):
        self.brand = brand
        self.model = model
        self.year = year

    def display_info(self):
        return f"{self.year} {self.brand} {self.model}"
```

1.2 What is an Object?

An **object** is an instance of a class. It holds real data and interacts with other objects.

Creating an Object from a Class

```
my_car = Car("Toyota", "Corolla", 2022)
print(my_car.display_info())  # Output: 2022 Toyota Corolla
```

---

2. Understanding the `__init__()` Method

The `__init__()` method is a special constructor method in Python used to initialize an object's attributes.

```python
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

p1 = Person("Alice", 25)
print(p1.name)  # Output: Alice
print(p1.age)   # Output: 25
```

Key Points About `__init__()`

- It is automatically called when an object is created.
- It initializes attributes of the object.
- It allows each object to have different attribute values.

---

3. Instance and Class Attributes

3.1 Instance Attributes

Instance attributes are unique to each object. They are defined inside the `__init__()` method.

```python
class Dog:
    def __init__(self, name, breed):
        self.name = name  # Instance attribute
        self.breed = breed

dog1 = Dog("Buddy", "Labrador")
dog2 = Dog("Max", "Beagle")

print(dog1.name)  # Output: Buddy
print(dog2.name)  # Output: Max
```

3.2 Class Attributes

Class attributes are shared across all instances of a class.

```python
class Animal:
    species = "Mammal"  # Class attribute

a1 = Animal()
```

```
a2 = Animal()

print(a1.species)  # Output: Mammal
print(a2.species)  # Output: Mammal
```

Difference Between Class and Instance Attributes

Feature	Instance Attribute	Class Attribute
Scope	Specific to an object	Shared across all instances
Defined in	`__init__()`	Outside any method
Modification	Only affects one instance	Affects all instances

---

4. Methods in a Class

4.1 Instance Methods

Instance methods work on individual objects.

```
class Student:
    def __init__(self, name, grade):
        self.name = name
        self.grade = grade

    def get_details(self):
        return f"{self.name} is in grade {self.grade}."

s1 = Student("John", 10)
print(s1.get_details())  # Output: John is in grade 10.
```

4.2 Class Methods (`@classmethod`)

Class methods work on the class rather than an instance.

```
class School:
    school_name = "Greenwood High"

    @classmethod
    def change_school_name(cls, new_name):
```

```
        cls.school_name = new_name

print(School.school_name)  # Output: Greenwood High
School.change_school_name("Sunrise Academy")
print(School.school_name)  # Output: Sunrise Academy
```

4.3 Static Methods (`@staticmethod`)

Static methods are independent of class and instance attributes.

```
class MathUtils:
    @staticmethod
    def add(a, b):
        return a + b

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

Type	Purpose	Uses `self`?	Uses `cls`?
Instance Method	Works with object attributes	✅	❌
Class Method	Works with class attributes	❌	✅
Static Method	Independent utility function	❌	❌

Here’s a more detailed introduction to Django Models to provide better clarity and depth.

---

5. Working with Objects in Django

Introduction to Django Models

In Django, **models** are Python classes that define the structure and behavior of database tables. Each model maps directly to a single table in the database, and Django provides an **Object-Relational Mapping (ORM)** system to interact with the database using Python instead of SQL.

A model typically includes:

- **Fields**: Attributes that define the data structure (e.g., `CharField`, `IntegerField`, `DateField`).
- **Methods**: Functions that operate on model instances.
- **Meta Options**: Configurations like ordering, database table name, etc.

Defining a Model in Django

To define a model, create a class that inherits from `models.Model` and specify fields as class attributes.

```
from django.db import models

class Book(models.Model):
    title = models.CharField(max_length=200)  # Title of the book
    author = models.CharField(max_length=100)  # Author name
    published_date = models.DateField()  # Date of publication
    price = models.DecimalField(max_digits=6, decimal_places=2, null=True,
blank=True)  # Optional price field

    def __str__(self):
        return self.title  # Returns the book title when printed

    def get_book_info(self):
        return f"{self.title} by {self.author}"
```

Explanation of Fields

- `CharField(max_length=...)` → Stores short text data (e.g., title, author).
- `DateField()` → Stores date values (e.g., publication date).
- `DecimalField(max_digits=6, decimal_places=2)` → Stores decimal values (e.g., price of the book).
- `null=True, blank=True` → Allows a field to be empty.

Applying Migrations

Once a model is defined, you need to create and apply migrations to reflect changes in the database.

```
python manage.py makemigrations
python manage.py migrate
```

Creating an Object in Django Shell

Django provides an interactive Python shell to work with models.

```
python manage.py shell
```

```
from myapp.models import Book

# Creating an object
book1 = Book(title="Django for Beginners", author="William S. Vincent",
published_date="2023-05-10", price=29.99)
book1.save()  # Saves the object to the database

# Retrieving the object
print(book1.get_book_info())  # Output: Django for Beginners by William S.
Vincent
```

Querying Objects

Once you have created objects, you can retrieve them using Django's ORM.

Retrieving All Objects

```
books = Book.objects.all()
for book in books:
    print(book.title)
```

Filtering Objects

```
books_by_author = Book.objects.filter(author="William S. Vincent")
```

Retrieving a Single Object

```
book = Book.objects.get(id=1)  # Retrieves the book with ID 1
```

Updating an Object

```
book.title = "Updated Django Book"
book.save()
```

Deleting an Object

```
book.delete()
```

6. Exercises

Exercise 1: Define a Class and Create an Object

Write a Python class called `Laptop` with attributes `brand`, `model`, and `price`.
Create an object of the class and print the details.

Exercise 2: Implement Class Methods

Modify the `Laptop` class to include a class attribute `category = "Electronics"`.
Add a class method to update the category and test it.