Class and Objects in Python

In Python, a class is a blueprint or a template for creating objects. Objects are instances of a class, and they encapsulate data (attributes) and behaviors (methods). The syntax for defining a class and creating objects in Python is as follows:

Class Definition:

```
class ClassName:
    # Class attributes (optional)

    def __init__(self, parameter1, parameter2, ...):
        # Constructor method (optional)
        self.attribute1 = parameter1
        self.attribute2 = parameter2
        # ...

    def method1(self, arg1, arg2, ...):
        # Method definition
        # Access class attributes using self.attribute_name
        # ...

    def method2(self, arg1, arg2, ...):
        # Another method definition
        # ...
```

Creating Objects:

```
# Creating an object of the class
object_name = ClassName(argument1, argument2, ...)

# Accessing attributes and methods of the object
object_name.attribute1
object_name.method1(argument1, argument2, ...)
```

Now, let's illustrate this with an example:

```
# Class definition
class Car:
    # Class attribute
    wheels = 4

    # Constructor method (initialize object attributes)
    def __init__(self, make, model, year):
        self.make = make
        self.model = model
        self.year = year
        self.is_running = False

    # Method to start the car's engine
    def start_engine(self):
        if not self.is_running:
            print(f"The {self.year} {self.make} {self.model}'s engine is now running.")
```

```
            self.is_running = True
        else:
            print("The engine is already running.")

    # Method to stop the car's engine
    def stop_engine(self):
        if self.is_running:
            print(f"The {self.year} {self.make} {self.model}'s engine is now stopped.")
            self.is_running = False
        else:
            print("The engine is already stopped.")

# Creating objects of the class
car1 = Car("Toyota", "Camry", 2020)
car2 = Car("Honda", "Accord", 2021)

# Accessing attributes and methods of the objects
print(f"Car 1: {car1.make} {car1.model} ({car1.year}) with {car1.wheels} wheels.")
car1.start_engine()
car1.stop_engine()

print("\n")

print(f"Car 2: {car2.make} {car2.model} ({car2.year}) with {car2.wheels} wheels.")
car2.start_engine()
```

```
car2.start_engine()
car2.stop_engine()
```

In this example, we defined a "Car" class with attributes (`make`, `model`, `year`, `is_running`) and methods (`start_engine`, `stop_engine`). We then created two instances of the class (`car1` and `car2`) and accessed their attributes and methods.

Key Points:

- **`__init__`:** The constructor method initializes the object's attributes when an instance is created.
- **`self`:** It refers to the instance of the class and is used to access instance attributes and methods.
- **Class attributes** (like `wheels`) are shared among all instances, while instance attributes (like `make`, `model`, `year`, `is_running`) are specific to each object.

This example demonstrates the basic structure of a class, the creation of objects, and how to access their attributes and methods in Python.