Class & Object in Python

MUKESH KUMAR

AGENDA

- ✓ Introduction to Classes & Objects
- ✓ Inbuilt Classes in Python
- ✓ Methods & Attributes
- ✓ Python Constructors
- ✓ Instance Variables & Class Variables
- ✓ Summary & Key Takeaways

What are Classes and Objects?

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

- An **object** is an instance of a class that has actual values assigned to its properties and can perform actions defined by its methods.

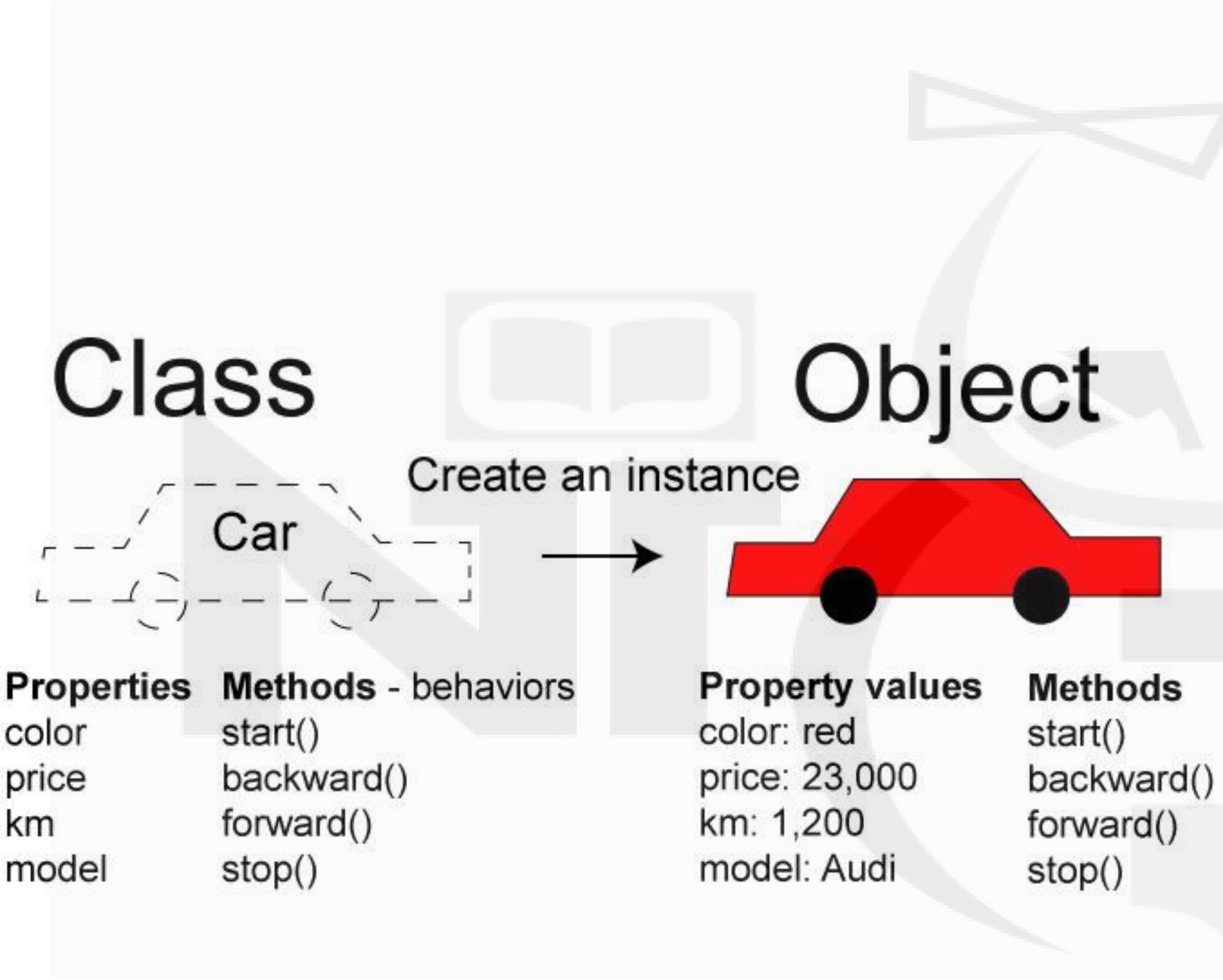
Real-world Analogy for Better Understanding

Consider a **Car Factory**:

- The **class** represents a blueprint for making cars.

- Each **car** produced from this blueprint is an **object.**

- Every car has common attributes (e.g., brand, model, color) but different values.



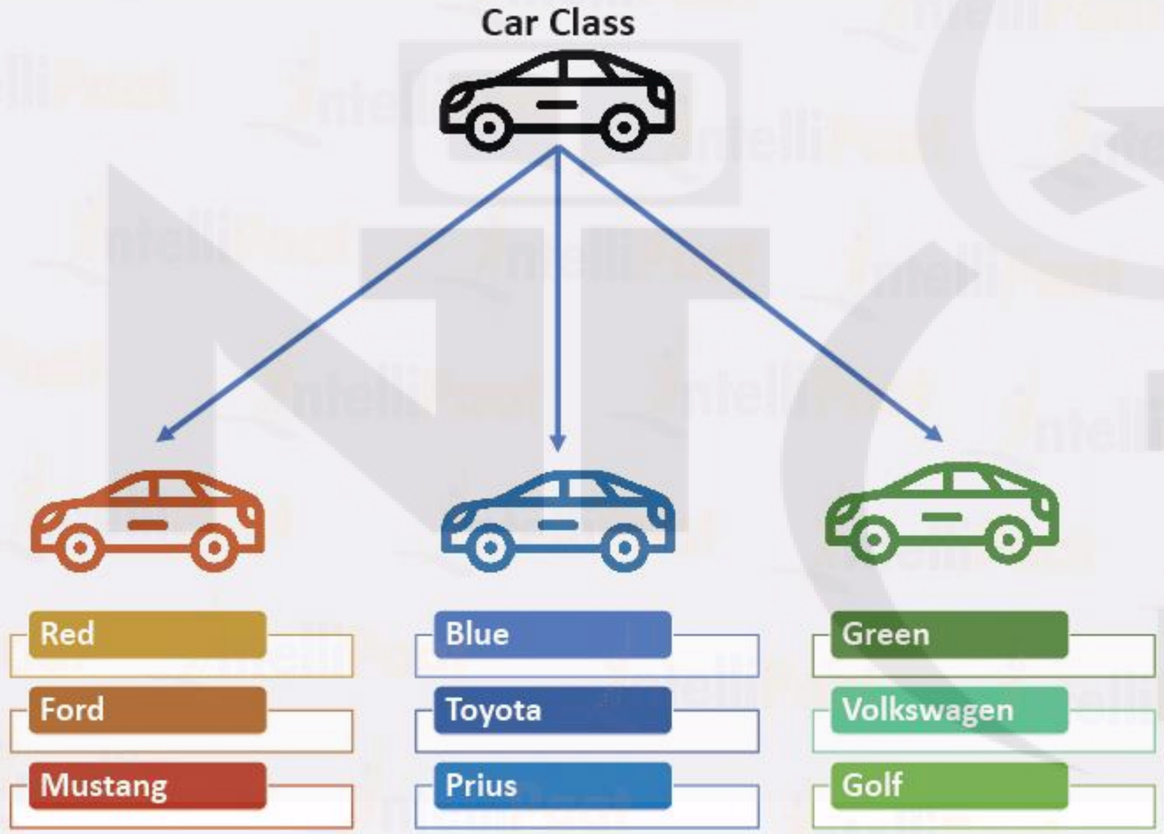
Class

Object

Create an instance

Properties	Methods - behaviors
color	start()
price	backward()
km	forward()
model	stop()

Property values	Methods
color: red	start()
price: 23,000	backward()
km: 1,200	forward()
model: Audi	stop()



Car Class

Red

Ford

Mustang

Blue

Toyota

Prius

Green

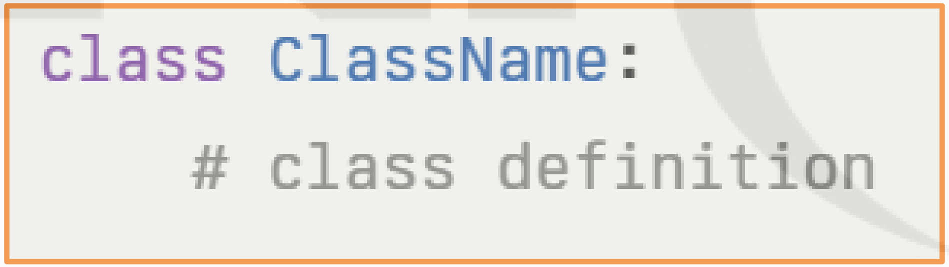
Volkswagen

Golf

How to Create a User Class?

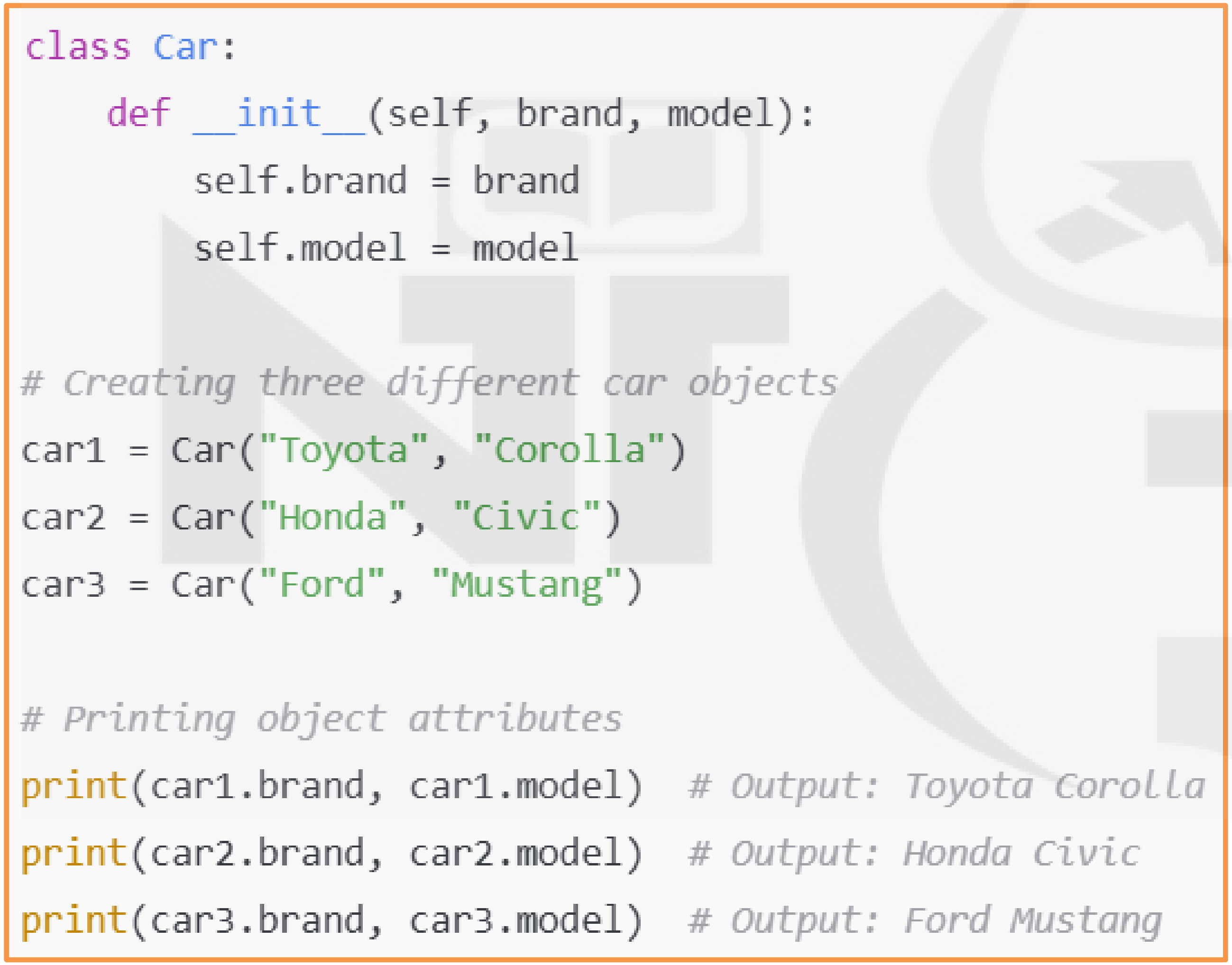
- A **User Class** in Python refers to a class that is defined by the programmer to model real-world entities or concepts.

- Syntax of class creation:



```
class ClassName:
    # class definition
```

Class Example



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

# Creating three different car objects
car1 = Car("Toyota", "Corolla")
car2 = Car("Honda", "Civic")
car3 = Car("Ford", "Mustang")

# Printing object attributes
print(car1.brand, car1.model)  # Output: Toyota Corolla
print(car2.brand, car2.model)  # Output: Honda Civic
print(car3.brand, car3.model)  # Output: Ford Mustang
```

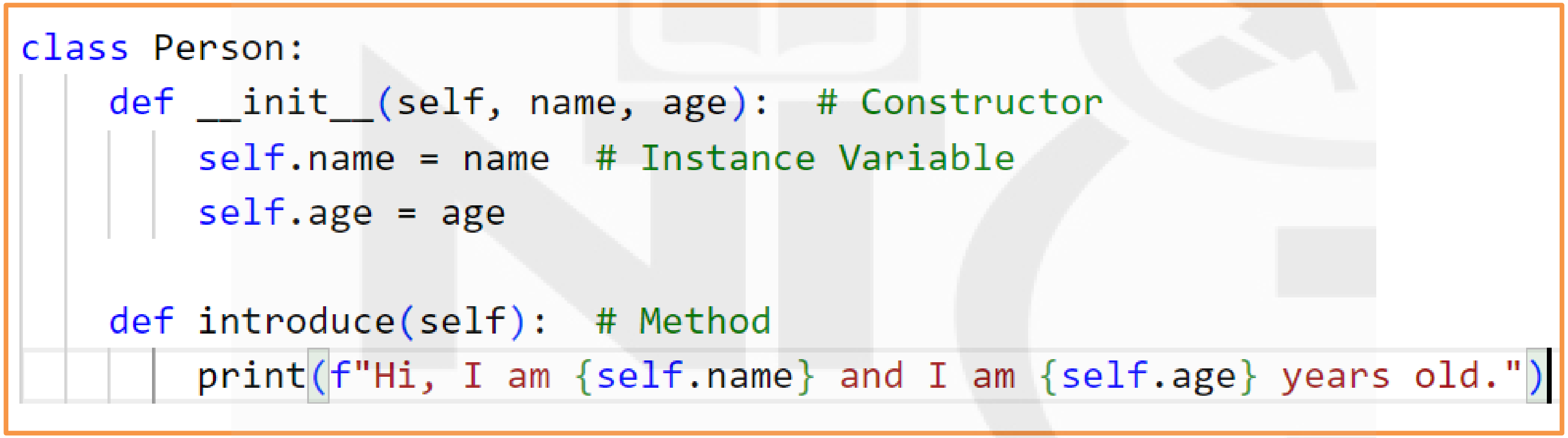
CREATING A CLASS & OBJECT

Structure of a class

A class consists of:

- **Class Definition:** Using the class keyword.
- **Attributes** (Instance Variables)**:** Data stored in an object.
- **Methods** (Functions within the Class)**:** Define behavior of the objects.
- **Constructor** (__init__ Method)**:** Special method to initialize instance variables.
- **Object Creation:** Instantiating an instance of the class.

Class Example

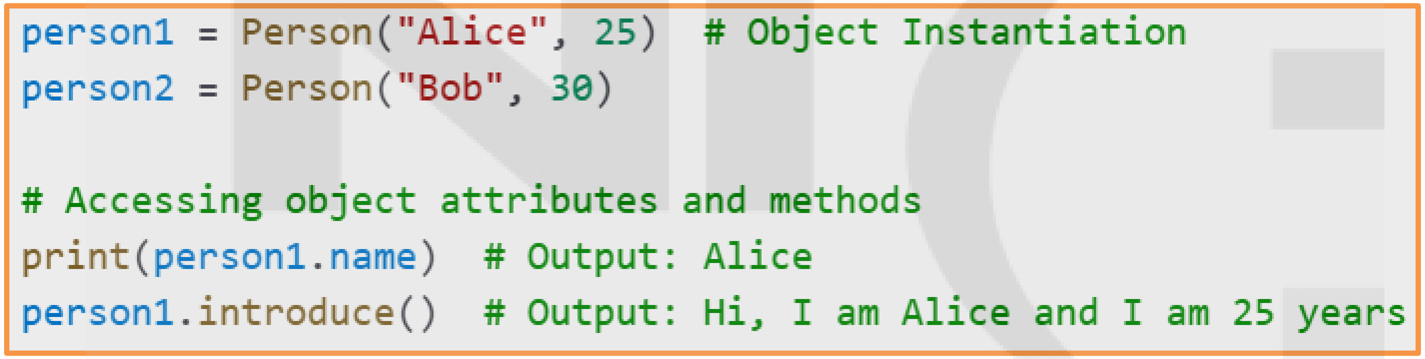


```python
class Person:
    def __init__(self, name, age):  # Constructor
        self.name = name  # Instance Variable
        self.age = age

    def introduce(self):  # Method
        print(f"Hi, I am {self.name} and I am {self.age} years old.")
```

Creating an Object of the Class

- To create an object, we use the class name followed by parentheses, passing required arguments (if any) to the constructor.



```python
person1 = Person("Alice", 25)  # Object Instantiation
person2 = Person("Bob", 30)

# Accessing object attributes and methods
print(person1.name)  # Output: Alice
person1.introduce()  # Output: Hi, I am Alice and I am 25 years
```

Key Points:

- Objects are created using the class name followed by parentheses.

- We can create multiple objects from a single class.

- Methods allow objects to perform actions.

- Attributes store individual object data.

Attributes and Methods

- **Attributes** are variables that hold data related to an object, representing its properties or characteristics

- **Methods** are functions associated with an object or class that perform specific actions on that object

- Methods and attributes are fundamental to object-oriented programming (OOP), allowing objects to have their own data and behaviors, which helps in modeling real-world entities

Attributes

- Attributes are variables that belong to an object and store information about its state.

  - They are defined within a class definition using the syntax **attr_name = attr_value**.

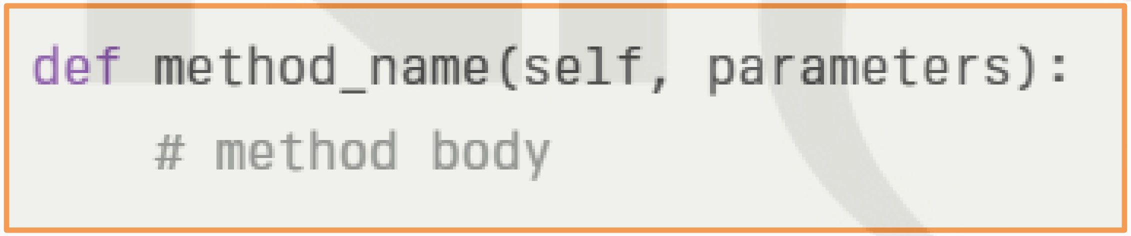
  - To access an attribute, you first create an instance of the class and then use the dot (.) operator: **instance_name.attr_name**.

  - Attributes can be **class attributes** (shared by all instances) or **instance attributes** (unique to each instance).

Methods

- Methods are functions defined within a class that define the actions an object can perform.

- They are defined within a class using the syntax:



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

  - where method_name is the name of the method, self refers to the instance of the object, and parameters are the arguments required by the method

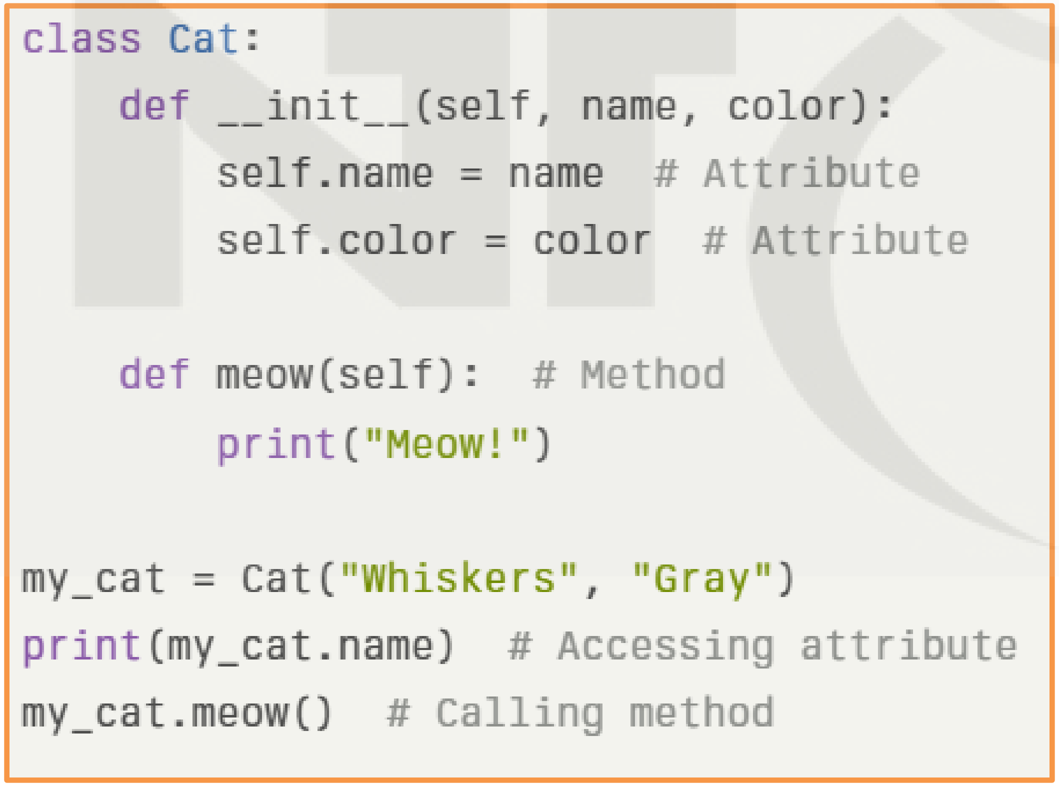
Methods

- To call a method, you use the dot (.) operator on an instance of the class: **instance_name.method_name()**.

- Each method belongs to an object, such that calling method(self) is equivalent to calling method(object)

Method Example

- In this example, **name** and **color** are attributes of the Cat class, while **meow()** is a method1.
- The attributes store the cat's name and color, and the method makes the cat meow



```
class Cat:
    def __init__(self, name, color):
        self.name = name  # Attribute
        self.color = color  # Attribute

    def meow(self):  # Method
        print("Meow!")

my_cat = Cat("Whiskers", "Gray")
print(my_cat.name)  # Accessing attribute
my_cat.meow()  # Calling method
```

What is Meant by Inbuilt Class?

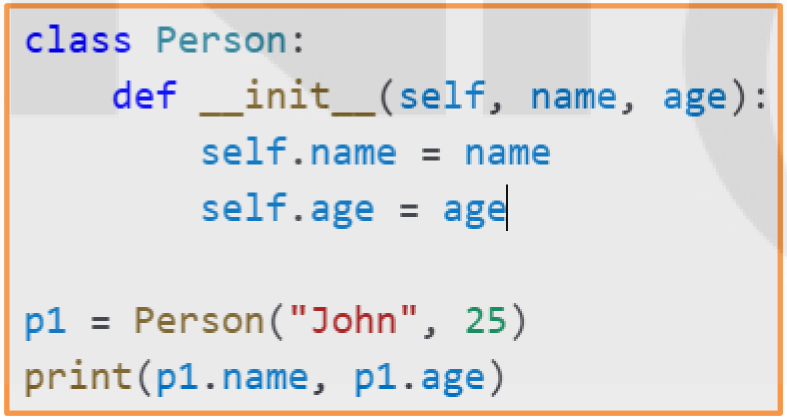
- Examples: list, dict, set, tuple, str, etc.

INIT() METHOD

Understanding __init__ Method

- It is called a constructor in Python

- Example



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

p1 = Person("John", 25)
print(p1.name, p1.age)
```

Python Constructor

- A constructor is a special method within a class that automatically initializes a new object when it is created.

- The constructor's main role is to assign values to the data members of the class.

- Python uses a method called __init__() to achieve this.

- **When a new object is created, the constructor is automatically called. If a class doesn't have a defined constructor, Python automatically creates a default constructor**

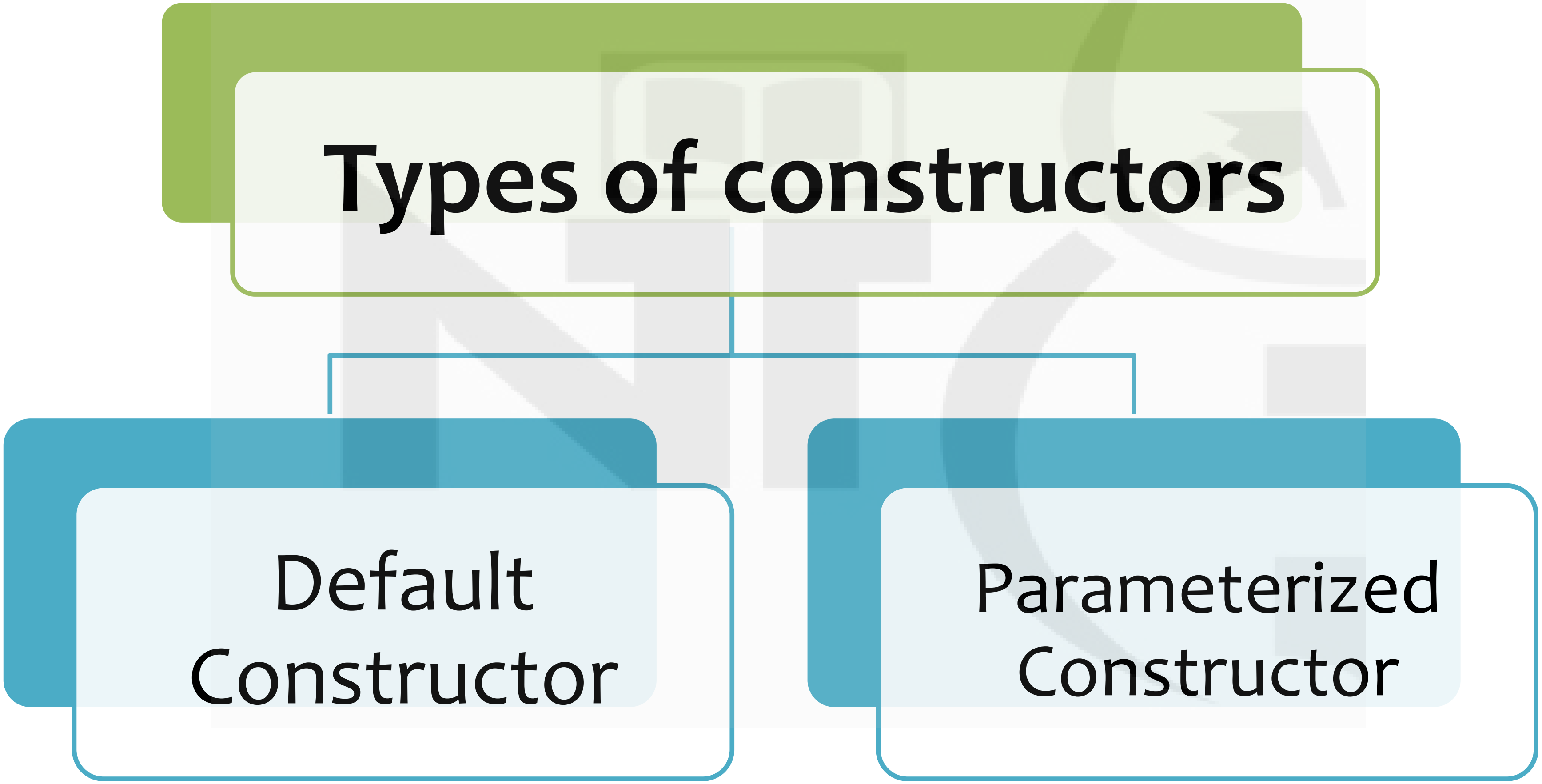
Rules of Python Constructor

- It starts with the **def** keyword, like all other functions in Python.

- It is followed by the word **init**, which is prefixed and suffixed with double underscores with a pair of brackets, i.e., **__init__().**

- It takes an argument called **self**, assigning values to the variables.

Constructor Types



Types of constructors

Default Constructor

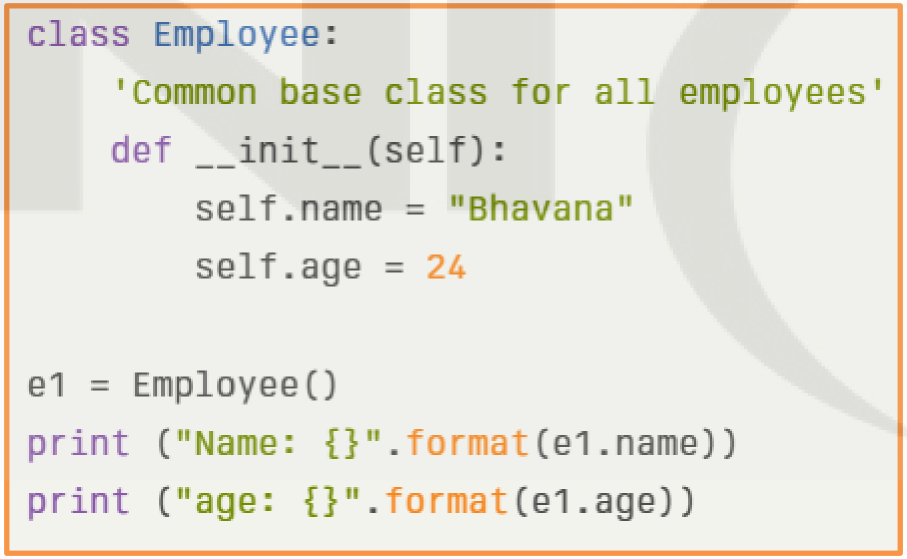
Parameterized Constructor

Default Constructor

- This constructor does not accept any arguments other than self.
- When a class doesn't explicitly define a constructor, Python provides a default one.

Default Constructor Example

- In this example, each object declared from the Employee class will have the same default values for the instance variables name and age.



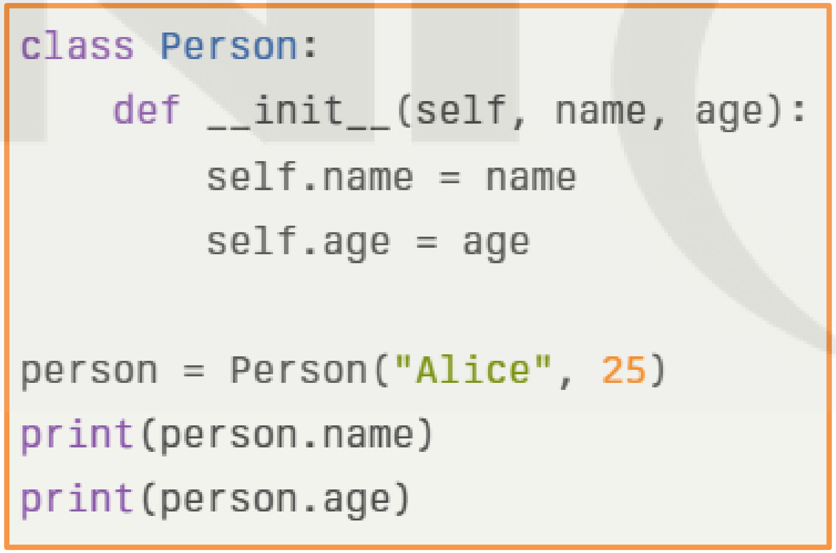
```python
class Employee:
    'Common base class for all employees'
    def __init__(self):
        self.name = "Bhavana"
        self.age = 24

e1 = Employee()
print ("Name: {}".format(e1.name))
print ("age: {}".format(e1.age))
```

Parameterized Constructor

- This constructor accepts one or more arguments, allowing instance variables to be initialized with specific values upon object creation.

- In this case, the __init__ method is a parameterized constructor that sets the name and age attributes of the Person object to the provided values



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

person = Person("Alice", 25)
print(person.name)
print(person.age)
```

SELF()

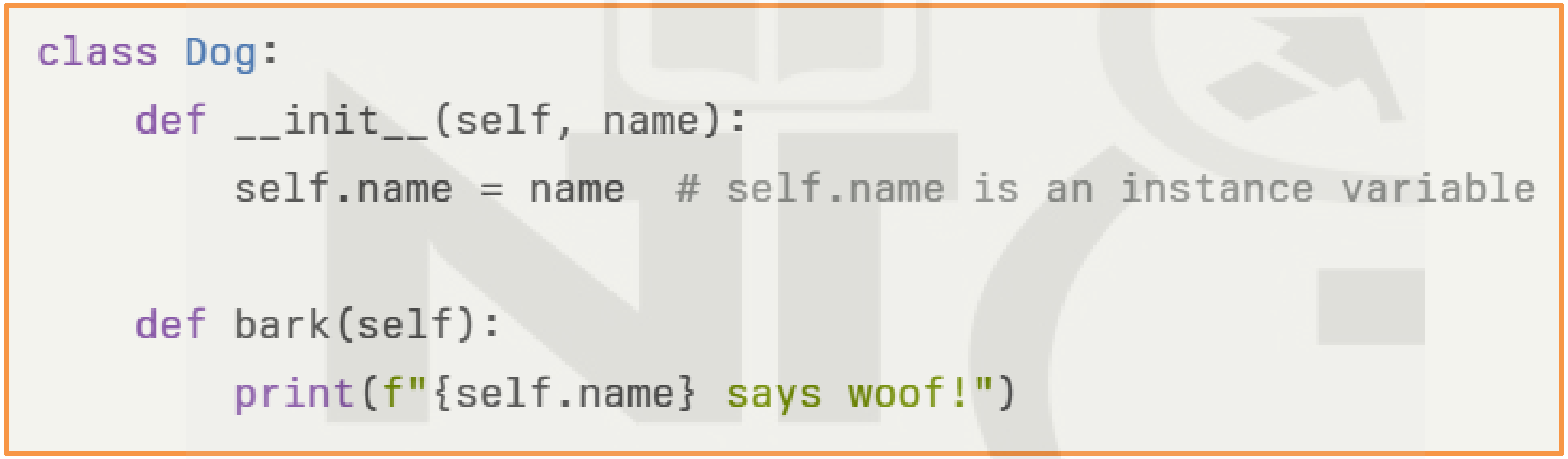
- In Python, **self** is a special parameter used in class methods to refer to the instance of the class.
- It allows you to access and modify the attributes and methods of that instance.
- When you define a method within a class, the first parameter is conventionally named self.
- Although you can technically use any name, sticking to self enhances readability and aligns with Python's coding conventions.

How self() works

- **Instance Reference:** self represents the instance of the class. When a method is called on an object, Python automatically passes the object itself as the first argument to the method.

- **Accessing Attributes:** You use self to access and modify instance variables. This distinguishes them from local variables.

Example



```python
class Dog:
    def __init__(self, name):
        self.name = name  # self.name is an instance variable

    def bark(self):
        print(f"{self.name} says woof!")
```

- In this case, **self.name** refers to the instance variable name, allowing each Dog instance to have its unique name

Why is self() needed

- Without self, methods wouldn't know which instance's data to operate on, leading to errors.

- self ensures that changes to an attribute affect the instance and not just a temporary variable.

Instance Variables

- Variables owned by the instances of a class, meaning each instance has its own copy.
- They are defined within methods, particularly the constructor (__init__).
- Instance variables are accessed using the instance name (e.g., instance_name.variable_name).

Instance Variables

- Changes to an instance variable only affect that specific instance and do not propagate to other instances.

- They are used to store data that is unique to each object.

Class Variables

- Variables defined within a class are shared among all instances of that class.

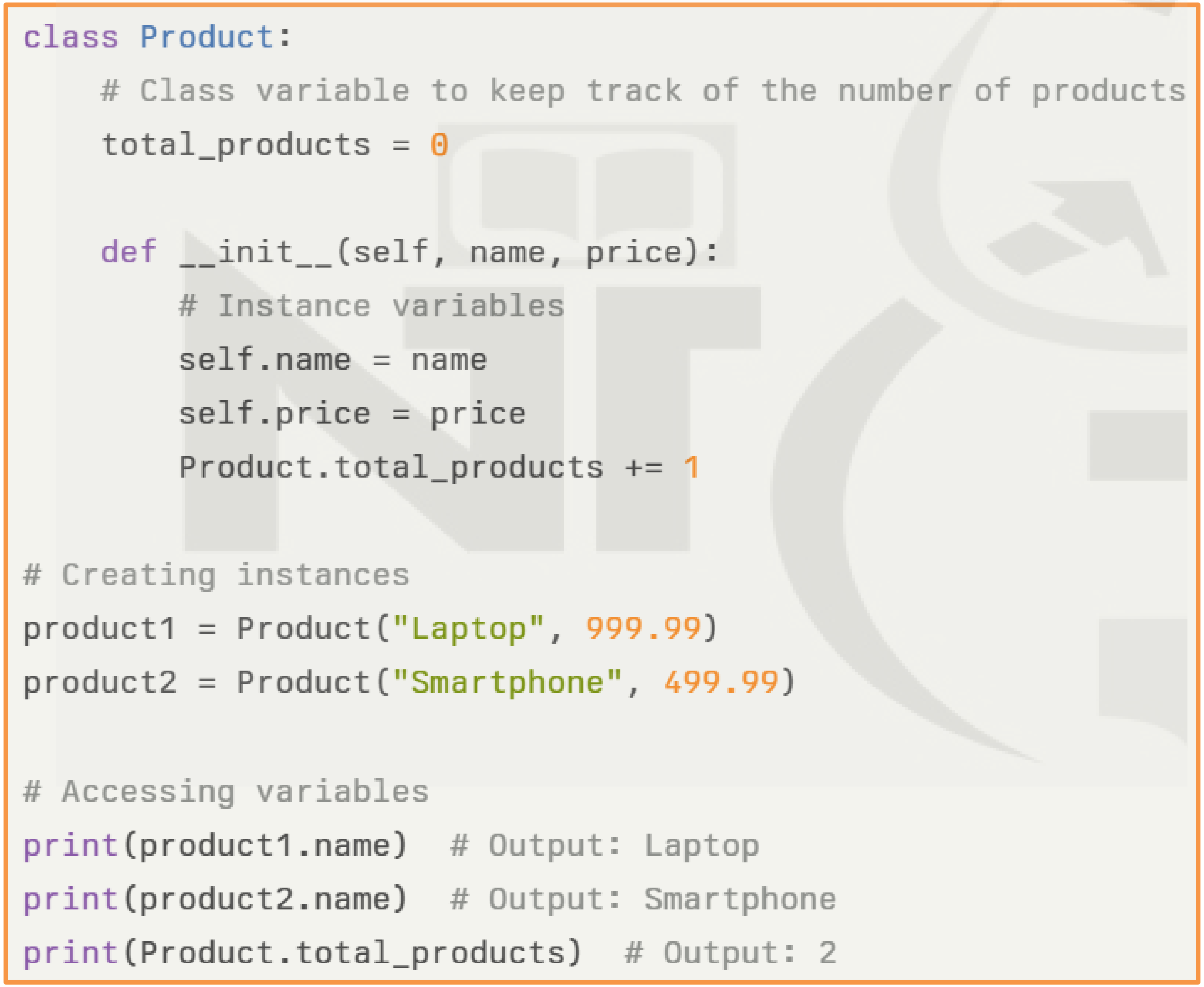
- They reside at the class level and are typically placed directly under the class header, before any methods.

- Class variables are accessed using the class name itself (e.g., ClassName.variable_name) or through an instance of the class (e.g., instance_name.variable_name).

Class Variables

- Modifying a class variable affects all instances of the class.
- They are useful for storing values that should be consistent across all instances or for initializing variables.
- They can also be used to keep track of the number of instances created.
- Class variables can be of any data type available in Python.

Example



```python
class Product:
    # Class variable to keep track of the number of products
    total_products = 0

    def __init__(self, name, price):
        # Instance variables
        self.name = name
        self.price = price
        Product.total_products += 1

# Creating instances
product1 = Product("Laptop", 999.99)
product2 = Product("Smartphone", 499.99)

# Accessing variables
print(product1.name)  # Output: Laptop
print(product2.name)  # Output: Smartphone
print(Product.total_products)  # Output: 2
```

Summary

- Recap of all topics
- Key takeaways